Parallele Bildberechnung in einem Netzwerk von Workstations

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

DISSERTATION

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
von

Dipl.-Inform. (FH) Marcus Roth
aus Kaiserslautern

Referenten der Arbeit:	Prof. Dr. h.c. Dr.-Ing. José L. Encarnação
	Prof. Dr. Dieter W. Fellner
Datum der Einreichung	30. September 2005
Datum der mündlichen Prüfung	18. November 2005

D17
Darmstädter Dissertation 2005

1 Einleitung

1.1 Motivation

Seit den ersten Anfängen der Computergrafik besteht eine Diskrepanz zwischen der visuellen Qualität, die ein Anwender erwartet und dem, was technisch machbar ist. Obwohl heute ein handelsüblicher PC in der Lage ist, dreidimensionale Modelle mit mehreren Millionen Polygonen interaktiv darzustellen, sind die Ergebnisse der Computergrafik immer noch sehr weit von realen Darstellungen entfernt. Aus diesem Grund wird nach wie vor intensiv an der Entwicklung besserer Algorithmen und leistungsfähigerer Hardware gearbeitet.

Für die Entwicklung von Mikroprozessoren sagt das Gesetz von Moor eine Verdoppelung der Geschwindigkeit alle 18 Monate voraus. Obwohl das Ende dieser Entwicklung schon häufig vorausgesagt wurde, ist es bis heute gültig. Bei der Entwicklung von Grafik-Hardware verlief die Entwicklung in den letzten Jahren etwa doppelt so schnell. Die Polygonleistung vervierfachte sich etwa alle 18 Monate. Eine doppelte oder vierfache Geschwindigkeit in 18 Monaten hilft jedoch nicht bei der Lösung aktueller Probleme. Wenn es beispielsweise für eine bestimmte Anwendung erforderlich ist, ein Modell mit 100 Millionen Polygonen interaktiv zu berechnen, ist es wenig hilfreich, zu wissen, dass dies in vier oder acht Jahren kein Problem mehr darstellt. Aus diesem Grund müssen auf der Basis verfügbarer Techniken Wege gefunden werden, um eine Steigerung der Performanz zu erreichen. Ein Ansatz, der in vielen Bereichen erfolgreich eingesetzt wird, ist die Ausnutzung von Parallelität.

In den 80er und 90er Jahren wurde dieses Prinzip erfolgreich für die Computergrafik in hochspezialisierten Grafikrechnern realisiert. Es wurden Rechner wie z.B. die SGI Infinite Reality [MBD97] mit mehreren Prozessoren und mehreren parallel arbeitenden Grafiksubsystemen entwickelt. Jedes Grafiksubsystem war wiederum aus vielen parallel arbeitenden Einheiten zusammengesetzt. Solche Rechner lieferten für die damalige Zeit eine sehr hohe Performanz, sie waren jedoch teuer und wurden nur in geringen Stückzahlen hergestellt.

Mitte der 90er Jahre begann sich die Entwicklung der Computergrafik zu verändern. Es wurde erstmals eine 3D-Hardware-Beschleunigung für Computerspiele zu günstigen Preisen angeboten. Die ersten dieser Grafikkarten konnten zwar in ihrer Leistungsfähigkeit nicht mit den hochspezialisierten Grafikrechnern von SGI oder Sun konkurrieren, jedoch entstand im Laufe der Jahre ein Massenmarkt, der für immer komplexere Computerspiele leistungsfähige Hardware forderte. Die hohen Stückzahlen und ein starker Wettbewerb zwischen den Anbietern führten dazu, dass heutige PCs eine höhere Grafikleistung aufweisen als die Spezialhardware der 90er Jahre.

Parallele Systeme lassen sich auf vielfältige Weise realisieren. Sie lassen sich aus speziell hierfür entwickelter Hardware oder aus Standardkomponenten zusammensetzen. Meist spielt bei der Entwicklung solcher Systeme nicht allein die Leistung, sondern das Verhältnis von Preis zu Leistung eine entscheidende Rolle. Dieses Verhältnis ist dann am günstigsten, wenn Standard-

komponenten verwendet werden, für die ein Massenmarkt existiert. Anfang der 90er Jahre wurde, vorwiegend im wissenschaftlichen Umfeld, begonnen, parallele Systeme auf der Basis vernetzter PCs aufzubauen. Dieser Ansatz hat sich als so erfolgreich erwiesen, dass heute mehr als 60% der 500 schnellsten Rechner der Welt als Cluster von Standardkomponenten aufgebaut sind.

Im direkten Vergleich zu klassischen Multiprozessor-Systemen bieten Cluster-Systeme bei der Kommunikation zwischen den Knoten nur geringe Bandbreiten und hohe Latenzen. Dafür verfügen Cluster-Systeme meist über schnellere Prozessoren als Multiprozessor-Systeme. Aufgrund dieser veränderten Voraussetzungen lassen sich Algorithmen, die in den letzten zehn bis 20 Jahren für eine Implementierung in Hardware oder für Multiprozessor-Systeme entwickelt wurden, nicht ohne weiteres auf Cluster-Systeme übertragen. Für eine effiziente Bildberechnung in einem Cluster sind neue Algorithmen und neue Werkzeuge erforderlich. Weit verbreitete Werkzeuge wie Open Inventor oder OpenGL Performer [RoH94] bieten keine Unterstützung für parallele Bildberechnung in einem Cluster.

Um das Potential PC-basierter Systeme für interaktive Echtzeitgrafik nutzbar zu machen, initiierte Dirk Reiners im Jahr 2000 das Projekt OpenSG [Rei02,VBR02]. Ziel von OpenSG ist es, eine frei verfügbare moderne Software-Basis für interaktive 3D-Anwendungen aus den Bereichen Virtual- und Augmented-Reality (VR/AR) bereitzustellen. Ziel der vorliegenden Arbeit ist es, OpenSG so zu erweitern, dass es als komfortable und leistungsfähige Basis zur parallelen Bildberechnung in einem Cluster verwendet werden kann. Da bisher noch kein Szenengraphensystem mit einer integrierten Unterstützung von Cluster-Systemen entwickelt wurde, müssen neue Lösungsansätze gefunden werden.

Betrachtet man die rasante Entwicklung und Verbreitung von Cluster-Systemen, so ist abzusehen, dass diese Systeme in Zukunft auch für die Berechnung interaktiver Echtzeitgrafik sehr stark an Bedeutung zunehmen werden. Der Erfolg dieser Systeme wird jedoch wesentlich von der Verfügbarkeit leistungsfähiger Software abhängen.

1.2 Aufgabenstellung und Zielsetzung

Im Rahmen dieser Arbeit wird ein Szenengraphensystem entwickelt, mit dessen Hilfe es möglich ist, interaktive dreidimensionale Bilder auf einem Cluster-System parallel zu berechnen. Das Szenengraphensystem soll als Basis für die Entwicklung von VR- und AR-Applikationen dienen. Es werden alle Arbeitsschritte betrachtet, die erforderlich sind, um eine Szenenbeschreibung parallel in eine dreidimensionale Darstellung umzuwandeln und auf beliebigen Display-Systemen anzuzeigen. Dies beinhaltet die Synchronisation des Szenengraphen im Cluster, die dynamische Lastverteilung, die Bildkomposition und die Datenübertragung zwischen den Knoten eines Clusters. Da die parallele Bildberechnung auf Cluster-Systemen ein aktives Forschungsfeld ist und auch in Zukunft neue Ansätze und neue Systemumgebungen zu erwarten sind, soll eine erweiterbare Systemarchitektur bereitgestellt werden, in die zukünftige Entwicklungen ohne Änderung des Basissystems integriert werden können.

1.3 Zusammenfassung der wichtigsten Ergebnisse

Alle Ergebnisse dieser Arbeit sind in die Weiterentwicklung des Szenengraphensystems OpenSG eingeflossen. OpenSG ist heute der einzige verfügbare Szenengraph mit einer umfangreichen Unterstützung für die Bildberechnung in Clustern. Dieses Alleinstellungsmerkmal hat viel zur Verbreitung von OpenSG beigetragen.

Die Unterstützung von Clustern ist vollständig in den Szenengraph integriert. Es wurden Protokolle entwickelt, um den Szenengraph sehr schnell innerhalb eines Clusters zu synchronisieren. Es wurde ein verlässliches Multicast-Protokoll entwickelt, mit dessen Hilfe es möglich ist, auch in sehr großen Clustern große Datenmengen schnell zu verteilen.

Durch den neu entwickelten, sehr flexiblen Ansatz bei der Ansteuerung von Projektionssystemen mit mehreren Projektoren ist es möglich, jedes beliebige Projektionssystem zu betreiben. Auf der Basis von OpenSG werden heute einfache Stereo-Projektionen, mehrere Cave-Projektionen [CSD93], Rundprojektionen und die HEyeWall – eine gekachelte Stereo-Projektion bestehend aus 48 Projektoren – durch Cluster betrieben. Diese Projektionssysteme lassen sich durch den Einsatz der Cluster-Technik wesentlich günstiger produzieren.

Neben der Unterstützung beliebiger Projektionssysteme wurde im Rahmen dieser Arbeit ein Verfahren entwickelt, um eine effiziente Lastverteilung in komplexen Projektionssystemen durchzuführen. Hiermit ist es möglich, beispielsweise in einer Cave den Bildberechnungsaufwand zwischen den Leinwänden gleichmäßig zu verteilen. Ist die Performanz nicht ausreichend, können auf einfache Weise weitere Rechner zur Bildberechnung herangezogen werden.

Um sehr große Szenen in interaktiven Bildwiederholraten anzuzeigen, wurde ein neues Bildkompositionsverfahren für die parallele Bildberechnung unabhängiger Teilszenen entwickelt. Damit ist es möglich, in einem Cluster, bestehend aus 32 PCs, Szenen mit mehreren hundert Millionen Polygonen interaktiv darzustellen. Das entwickelte Bildkompositionsverfahren verringert die erforderliche Netzwerkbandbreite im Vergleich zu bestehenden Verfahren so weit, dass hohe Bildwiederholraten auch mit Standard-Netzwerkhardware erreicht werden. Zusätzlich wurde eine Lösung für die Darstellung transparenter Objekte bei der parallelen Bildberechnung von Teilszenen gefunden.

Es wurden Techniken entwickelt, um Szenen darzustellen, die nicht mehr vollständig von einem einzelnen Rechner geladen werden können. Mit Hilfe dieser Techniken kann das Laden großer Szenen auf viele Rechner verteilt werden. Neben dem geringeren Speicherverbrauch auf jedem einzelnen Rechner wird auch die Ladegeschwindigkeit stark beschleunigt.

Alle Verfahren wurden in das Szenengraphensystem OpenSG integriert. Durch die vollständige und nahtlose Integration können Anwendungen, die auf OpenSG basieren, sehr einfach und effizient in einem Cluster eingesetzt werden.

2 Grundlagen

Dieses Kapitel beschreibt den Stand der Technik zur parallelen Bildberechnung auf vernetzten Workstations und die für das Verständnis erforderlichen Grundlagen. Genauere Analysen und Vergleiche, die im Rahmen dieser Arbeit durchgeführt wurden, werden in späteren Kapiteln ausführlich beschrieben.

2.1 Cluster-Systeme

In einem klassischen Multiprozessor-System werden mehrere Prozessoren über schnelle Datenpfade untereinander und mit einem gemeinsamen Hauptspeicher verbunden. Auf dieser Systemarchitektur aufbauend, wurden in den 80er und 90er Jahren des 20. Jahrhunderts sehr leistungsfähige, aber auch extrem teure Rechnersysteme gebaut. Aufgrund der hohen Kosten waren solche Hochleistungsrechner nur in den Bereichen einsetzbar, in denen Leistung ohne Rücksicht auf die Kosten gefordert wurde.

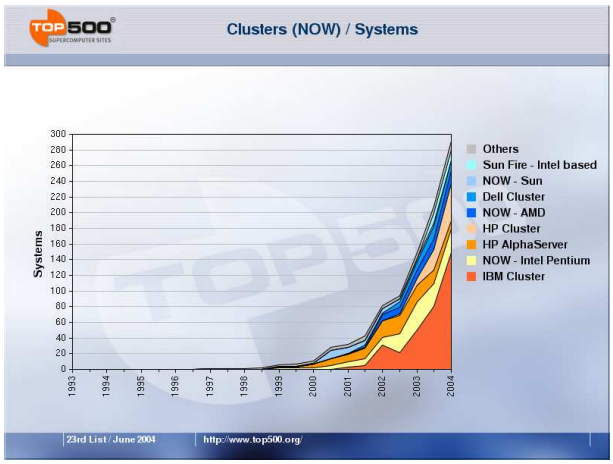

Abbildung 2-1: Cluster in der Top-500-Liste (Quelle Top-500.org)

Klassische massiv parallele Multiprozessor-Systeme haben in den letzen Jahren stark an Bedeutung verloren und sind von einem neuen wesentlich pragmatischeren Ansatz verdrängt worden. Die Idee bestand darin, bewusst Abstriche bei der Leistung in Kauf zu nehmen und dafür kostengünstige Standard-Computer zu vernetzen und als ein paralleles Computersystem zu betreiben. Solche Systeme werden als Commodity-Cluster oder auch COTS (Commodity of the shelf) oder auch als NOWs (Network of Workstations) bezeichnet. Das erste Cluster-System, das einen größeren Bekanntheitsgrad erlangt hat, ist der BEOWOLF-Cluster [BSS95]. Er bildete die Grundlage für viele weitere Cluster-Systeme an Universitäten, von denen einige den Sprung in die Liste der 500 schnellsten Rechner der Welt schafften. Meist werden Cluster-Systeme mit Hilfe von Standard-Betriebssystemen wie z.B. Linux, OS-X oder Windows aufgebaut. Es bestehen jedoch auch Ansätze, einen Cluster von einem Betriebssystem als Einheit zu verwalten. Ein Beispiel hierfür ist das auf Linux basierende MOSIX-Projekt [LaL98].

Die Erfahrung mit parallelen Systemen hat gezeigt, dass hohe Datenraten und sehr kurze Latenzen bei der Kommunikation sehr wichtige Faktoren für eine effiziente Parallelverarbeitung sind. Der Erfolg von Cluster-Systemen ist erstaunlich, denn sie bieten weder hohe Datenraten noch geringe Latenzen. Standard-Netzwerke wie z.B. Ethernet haben durch den komplexen Protokoll-Stack sehr hohe Latenzen, und die Datenraten sind wesentlich geringer als in einem eng gekoppelten Multiprozessorsystem. Cluster-Systeme bieten keinen schnellen Zugriff auf einen gemeinsamen Speicher, was die Software-Entwicklung erschwert. Die Erfahrung der letzten Jahre hat gezeigt, dass die Nachteile von Cluster-Systemen sicher für einige Anwendungsgebiete nicht akzeptabel sind. Aber es hat sich auch gezeigt, dass für viele Anwendungsgebiete der günstige Preis die Nachteile aufwiegt. Viele Anwendungen wurden erfolgreich dahingehend optimiert, dass sie weniger abhängig von kurzen Latenzen und hohen Datenraten sind. Für rechenintensive Anwendungen lässt sich auf der Basis eines Cluster-Systems die gleiche Prozessorleistung zu wesentlich günstigeren Preisen realisieren als mit einem Multiprozessor-System.

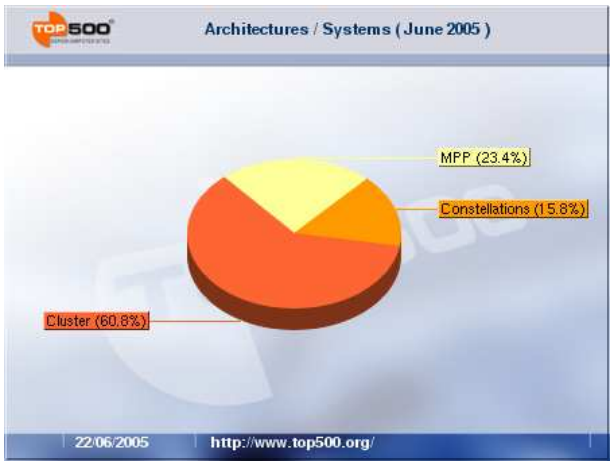

Abbildung 2-2: Anteil der Cluster in der Top-500-Liste (Quelle Top-500.org)

Der Bau eines Cluster-Systems erfordert keinen langwierigen Entwicklungsprozess. Es ist möglich, innerhalb eines Tages die benötigten Einzelteile einzukaufen und zu einem funktionierenden parallelen Computer-System zusammenzusetzen. Dies ist nicht nur ein Kostenvorteil, sondern bietet auch die Möglichkeit, sehr schnell auf technische Entwicklungen zu reagieren. Auf dem Gebiet der Computergrafik wird von den führenden Herstellern etwa jedes halbe Jahr eine neue Produktgeneration auf den Markt gebracht. Besitzer eines klassischen Multiprozessor-Grafikrechners von SGI können von dieser Entwicklung nicht profitieren. Ein Cluster-System kann jedoch in wenigen Stunden mit aktuellen Grafikkarten ausgestattet werden. Ein weiterer Vorteil besteht darin, dass ein Cluster-System beliebig um weitere Rechner erweitert oder teilweise mit neuen Rechnern ausgestattet werden kann.

Betrachtet man nur die CPU-Leistung, so ist das Preis-Leistungs-Verhältnis eines Clusters wesentlich günstiger als das eines Multprozessor-Systems. Ein Cluster-System kann jedoch nur dann sinnvoll eingesetzt werden, wenn es gelingt, ein Problem trotz der geringen Bandbreite und der hohen Latenz effizient zu parallelisieren. Bestehende und bewährte Verfahren, die für

Multiprozessorsysteme entwickelt wurden, sind nicht ohne weiteres auf einen Cluster zu übertragen. Eine sehr umfassende und weiterführende Beschreibung von Cluster-Systemen findet sich in [Bak00].

2.2 Datenverteilungsstrategien

Bei der parallelen Berechnung von 3D-Szenen in einem Cluster müssen die einzelnen Knoten des Clusters so miteinander synchronisiert werden, dass alle Knoten gemeinsam aus der gleichen Datenbasis ein konsistentes Bild berechnen. In einem verteilten System müssen Informationseinheiten zwischen den parallelen Instanzen ausgetauscht werden. Bei der Untersuchung bestehender Ansätze können unterschiedliche Aspekte betrachtet werden. Für jedes verteilte System muss geklärt werden, welche Informationseinheiten zur Synchronisation verwendet werden und wie diese Informationseinheiten zwischen den verteilten Instanzen übertragen werden.

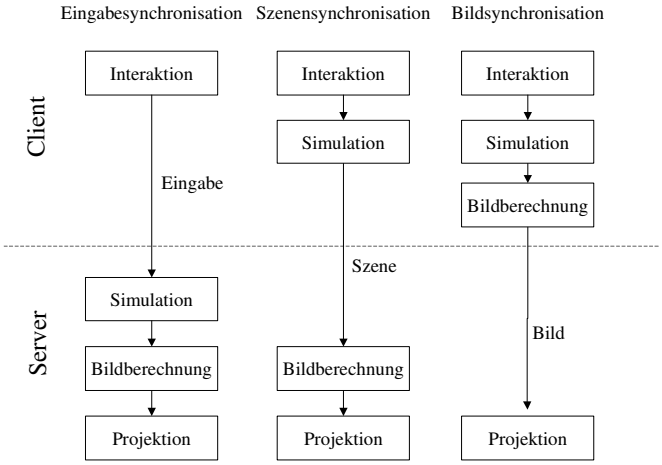

Abbildung 2-3: Synchronisationsstrategien

M. R. Macedonia und M. J. Zyda [MaZ97] haben die Kommunikation in verteilten virtuellen Umgebungen untersucht. Die hierbei eingesetzten Synchronisationsmechanismen können auch für parallele Bildberechnungsverfahren in Clustern eingesetzt werden. Macedonia und Zyda untersuchen Kommunikationsmechanismen anhand von drei einfachen Fragen: Was wird verteilt, wie wird es verteilt, und warum wird es verteilt? Wichtige Faktoren bei der Kommunikation sind die Art der Übertragung, die Verzögerungszeit, die Verlässlichkeit der Übertragung und die Kommunikations-Bandbreite.

Im Folgenden werden unterschiedliche Verteilungsstrategien für die Bildberechnung aufgeführt. Es wird gezeigt, dass je nach Zeitpunkt der Synchronisation unterschiedliche Informationseinheiten zwischen den parallel arbeitenden Instanzen ausgetauscht werden müssen.

In einer interaktiven Applikation werden Eingaben eines oder mehrerer Benutzer durch Interaktionsgeräte durchgeführt. Ein Simulationssystem modifiziert aufgrund der Benutzereingaben

die interne Beschreibung der Szene. Anschließend wird aus der Beschreibung der Szene ein Bild synthetisiert und auf einem Monitor oder einer Projektionsleinwand angezeigt.

Wird die Verarbeitung einer solchen Applikation parallelisiert, müssen die parallel ablaufenden Programmteile auf eine gemeinsame und konsistente Datenbasis zugreifen. Han Chen, Yuqun Chen u.a. haben in [CCF01] eine Taxonomie für parallele Applikationen zum Betrieb von hochauflösenden Projektionswänden entwickelt. Diese Taxonomie lässt sich auch auf andere parallele Bildberechnungsverfahren anwenden. Eine Applikation wird zur Parallelisierung in einen Client- und einen Server-Teil aufgeteilt. Der Client-Teil ist nicht parallelisiert, während mehrere Instanzen des Server-Teils parallel ausgeführt werden. Abbildung 2-3 zeigt drei grundlegende Formen der Daten-Synchronisation.

- Eingabesynchronisation: Auf jedem Rechner im Cluster wird die gleiche Applikation ausgeführt. Die Eingaben des Anwenders werden an alle Instanzen der Applikation gesendet. Unter der Voraussetzung, dass der interne Zustand einer Applikation nur durch externe Eingaben bestimmt ist, sind alle parallelen Instanzen in einem identischen Zustand, solange sie alle Eingaben in der gleichen Reihenfolge empfangen. Diese Form der Synchronisation erfordert einen sehr geringen Kommunikationsaufwand. Umsetzungen finden sich u.a. in [CCC01]. Die Eingabesynchronisation wird häufig auch in fehlertoleranten Systemen verwendet, um mehrfach vorhandene Instanzen konsistent zu halten [Bre98, Tor00]. Auch in kollaborativen VR-Systemen werden häufig nur Ereignisse, die durch Eingaben ausgelöst wurden, synchronisiert [HaW98].

- Szenensynchronisation: Hierbei wird die Beschreibung der 3D-Szene zwischen den Instanzen synchronisiert. Es können sowohl Objekte des Szenengraphen oder grafische Primitive synchronisiert werden. Beispiele für eine Synchronisation des Szenengraphen, basierend auf Open Inventor, werden in [HSF99, ZHC00] beschrieben. Beispiele für die Verteilung von grafischen Primitiven sind WireGL [HEB01] und dessen Nachfolger Chromium [HHN02].

- Bildsynchronisation: Bei dieser Art der Synchronisation wird lediglich die Anzeige parallelisiert und synchronisiert. Die Client-Applikation führt die komplette Bildberechnung durch. Das fertige Bild wird anschließend aufgeteilt und über ein Netzwerk an die Projektionsrechner gesendet.

Han Chen, Yuqun Chen u.a. zeigen in [CCF01], dass die Auswahl des Synchronisationsverfahrens von den Eigenschaften der Applikation abhängig ist. Soll beispielsweise ein Partikelsystem visualisiert werden, so muss bei der Eingabesynchronisation lediglich die Zeit zwischen den parallel betriebenen Partikelsimulationssystemen synchronisiert werden. Bei der Szenensynchronisation müssen hingegen die Positionen aller Partikel übertragen werden, was einen wesentlich höheren Kommunikationsaufwand bedeutet. Auf der anderen Seite muss jedoch bei der

Eingabesynchronisation die Simulation von allen parallelen Instanzen durchgeführt werden. Dies führt dazu, dass die vorhandenen Ressourcen nur unzureichend ausgenutzt werden.

Für die parallele Bildberechnung ist die Bildsynchronisation ungeeignet, da hierbei nur die Darstellung parallelisiert wird. Die Bildberechnung erfolgt sequentiell. Aus diesem Grund werden zur Synchronisation meist Eingabe- oder Szenesynchronisation eingesetzt. Häufig werden je nach Anwendung auch Mischformen verwendet.

Unabhängig davon, welche Form der Synchronisation eingesetzt wird, ist ein schnelles Kommunikationsmedium unerlässlich. Der folgende Abschnitt gibt einen kurzen Überblick über weit verbreitete Netzwerksysteme für Cluster.

2.3 Netzwerkhardware für Cluster

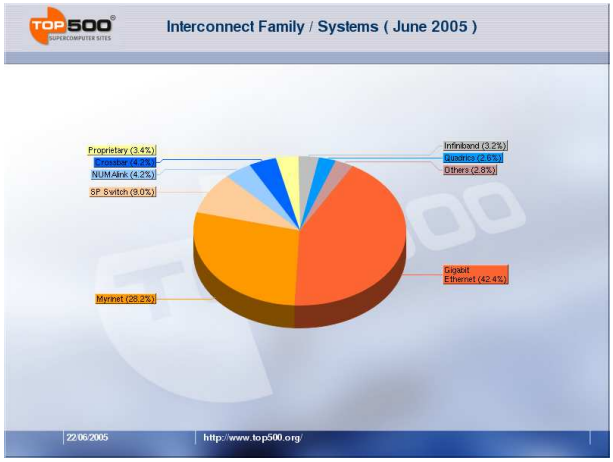

Abbildung 2-4: Netzwerkhardware in der Top-500-Liste (Quelle: Top500.org)

Das Hauptproblem bei der Umsetzung von Algorithmen auf einem Cluster besteht in der geringen Datentransferrate zwischen den einzelnen Rechnern. Es ist naheliegend, hierfür sehr schnelle Hardware zu entwickeln. Allerdings ist der Markt für Cluster-System im Vergleich zum PC-Markt sehr klein. Kosten für die Entwicklung von Spezialhardware können meist nicht durch eine hohe Stückzahl ausgeglichen werden. Aus diesem Grund findet man in vielen Clustern Standardkomponenten wie beispielsweise Ethernet. Als spezielle Cluster-Lösung konnte in den letzten Jahren lediglich Myrinet eine größere Verbreitung erreichen. Einige Hersteller von Server-Systemen haben sich zusammengeschlossen, um Infiniband als neues standardisiertes Verbindungsnetzwerk für Systemkomponenten zu entwickeln. Infiniband hat bisher jedoch noch keine weitere Verbreitung gefunden. Es ist allerdings sehr wahrscheinlich, dass Infiniband in Zukunft eine wichtige Rolle bei der Vernetzung von Cluster-Systemen spielen wird.

Bei der Auswahl einer geeigneten Netzwerktechnik für ein Cluster-System spielt meist der Preis eine wesentliche Rolle. Kurmann, Rauch und Stricker untersuchen in [KSR03] das Verhältnis von Preis zu Leistung für unterschiedliche Cluster-Netzwerke. Die Kosten reichen von etwa 4% bis zu 35% der Kosten eines Knotens. Die Verteilung der Kosten auf Switch, Verkabelung und Netzwerk-

Iinterface ist bei den untersuchten Systemen sehr unterschiedlich. Ein weiterführender Vergleich unterschiedlicher Systeme ist beispielsweise in [PLG99] zu finden.

Verbindung	Maximale Transferrate
Fast Ethernet	10 Mbyte/s
Gigabit Ethernet	100 Mbyte/s
10 Gigabit Ethernet	1 Gbyte/s
USB 2.0	60 Mbytes/s
FireWire 400 IEEE 1394	50 Mbyte/s
FireWire 800 IEEE 1394	100 Mbyte/s
FireWire 1600 IEEE 1394	200 Mbyte/s
Infiniband 1X	250 Mbyte/s
Infiniband 4X	1 Gbyte/s
Infiniband 12X	3 Gbyte/s
Myrinet	250 Mbyte/s
HyperTransport	8 Gbyte/s
PCI-Express x16	4 Gbyte/s

Tabelle 1: Cluster-Netzwerke

Die folgenden Abschnitte gehen auf weit verbreitete oder in Zukunft relevante Netzwerksysteme für Cluster ein. In der Literatur sind viele weitere Ansätze zu finden [SiJ02,PFH02], deren zukünftige Bedeutung jedoch nicht abzusehen ist.

2.3.1 Myrinet

Wie in Abbildung 2-4 zu sehen, ist Myrinet [BCF95], abgesehen von Ethernet, die am weitesten verbreitete Netzwerktechnik für Cluster. Es bietet eine Datenübertragungsrate von etwa 250 MByte/s und eine sehr kurze Latenzzeit. Myrinet-Netzwerke bestehen aus Vollduplex-Punkt-zu-Punkt-Verbindungen. Es werden Verbindungen zwischen Host und Switch und Verbindungen zwischen Switches aufgebaut. Myrinet verwendet eine paketorientierte Datenübertragung. Bei den Switches handelt es sich um Crossbar-Switches. Myrinet kann per Software über ein sehr schlankes proprietäres Interface angesprochen werden. Zusätzlich besteht die Möglichkeit, einen TCP/IP- und UDP/IP-Protokoll-Stack zu verwenden. Das Myrinet-Interface erlaubt es, Daten ohne zusätzlichen Kopiervorgang direkt zu versenden. Dies ist bei Standard-TCP/UDP nicht möglich.

2.3.2 Infiniband

Infiniband wurde als Netzwerk zur Anbindung von IO-Einheiten an die CPU und den Hauptspeicher entwickelt. Die Spezifikation 1.0 wurde im Jahr 2000 veröffentlicht. Die aktuelle Spezifikation 1.1 wurde 2002 veröffentlicht. Infiniband konnte sich bis heute noch nicht auf breiter

Front durchsetzen. Obwohl viele Firmen offiziell Infiniband unterstützen, werden bisher nur wenige darauf aufbauende Produkte angeboten.

Infiniband unterstützt verschiedene Geschwindigkeitsstufen: 2,5 GBit/s, 10 GBit/s und 30 GBit/s. Es werden serielle Punkt-zu-Punkt-Verbindungen mit einer sehr geringen Latenz eingesetzt. Sollte sich Infiniband im Server-Bereich durchsetzen, ist es sicher eine gute Alternative zu Myrinet. Multicast und Broadcast werden jedoch nicht unterstützt. Infiniband könnte eine sehr gute Wahl für die Vernetzung von Cluster-Systemen darstellen. Allerdings ist Infiniband bis heute nur wenig verbreitet. Dies kann sich jedoch sehr schnell mit der Verfügbarkeit günstiger Hardware ändern.

2.3.3 Ethernet

Ende 1972 entwickelte Robert Metcalfe bei Xerox ein experimentelles Netzwerk. Alle Stationen des Netzwerks übertrugen ihre Daten über einen gemeinsamen Bus. 1993 entstand der Name Ethernet. 1976 veröffentlichten Robert Metcalfe und David Boggs die Grundlagen von Ethernet „Ethernet: Distributed Packet Switching for Local Computer Networks“ [MeB76]. Ethernet wurde im Laufe der folgenden Jahre so erfolgreich, dass Xerox zusammen mit Intel und DEC den Quasi-Standard DIX-Ethernet veröffentlichte. 1983 wurde der Standard IEEE 802.3 veröffentlicht. Seit dieser Zeit wird der Standard kontinuierlich weiterentwickelt. 1995 wurde der Fast Ethernet Standard verabschiedet, 1998 Gigabit Ethernet und im Jahr 2002 10 Gigabit Ethernet.

Während die Geschwindigkeit von 10 MBit über 100 MBit, 1 GBit und jetzt 10 GBit kontinuierlich anstieg, fiel der Preis für Ethernet-Technik kontinuierlich. Mittlerweile werden kaum noch Computer angeboten, die nicht mindestens einen 100-MBit-Ethernet-Anschluss besitzen. Ethernet ist heute die dominierende Technik in lokalen Netzwerken.

Ethernet-Technik ist preiswert und für jede Hardware- und Software-Plattform erhältlich. Es ist daher naheliegend, auf dieser Basis Cluster-Systeme miteinander zu vernetzen. Ethernet bietet zudem die Möglichkeit, per Broadcast und Multicast Daten von einem Sender parallel an mehrere Empfänger zu versenden. Dies ermöglicht es, große Datenbestände zwischen den Rechnern eines Clusters zu synchronisieren. In reinen Punkt-zu-Punkt-Netzwerken wie beispielsweise Myrinet [BCF95] oder Infiniband [Inf02] müssen die Daten vom Sender einzeln zu jedem Empfänger gesendet werden.

Der Hauptgrund, warum spezielle Entwicklungen wie beispielsweise Myrinet häufig eingesetzt werden, liegt meist nicht an einer höheren Datenübertragungsrate, sondern an der hohen Latenz, die Ethernet mit dem darauf aufbauenden Protokoll-Stack aufweist. Die Funktionen von Ethernet werden üblicherweise über das Socket-Interface angesprochen. Dieses Interface ist mittlerweile als Posix-Standard verfügbar, und es ist Bestandteil aller aktuellen Betriebssysteme. Bei der Kommunikation wird eine verbindungsorientierte und eine paketorientierte Kommunikation unterstützt.

Die verbindungsorientierte Kommunikation wird mit Hilfe des TCP-Protokolls realisiert. TCP gewährleistet eine fehlerlose Datenübertragung zwischen zwei Endpunkten. Feste Punkt-zu-Punkt-Verbindungen werden auch als Stream-Sockets bezeichnet. Multicast und Broadcast sind hierbei nicht möglich. Die Datenübertragung ist sehr effizient, da Daten gepuffert und von Betriebssystem parallel zur Abarbeitung einer Applikation gesendet und empfangen werden.

Die paketorientierte Kommunikation wird mit Hilfe des UDP-Protokolls bereitgestellt. Es wird keine explizite Verbindung zwischen Sender und Empfänger aufgebaut. Jedes Nachrichtenpaket erhält eine Ziel-Adresse. Broadcast und Multicast werden über reservierte Adressbereiche angesprochen. Die Kommunikation ist theoretisch schneller als bei TCP, da der Protokoll-Overhead für die sichere Kommunikation entfällt. Allerdings macht sich dieser Vorteil nur dann bemerkbar, wenn für die Applikation eine sichere Datenübertragung nicht erforderlich ist. Beispielsweise ist bei der Übertragung von Video-Daten eine hohe Bildwiederholrate sehr wichtig. Fehlende Bildteile sind meist nicht so störend wie eine sehr geringe Bildrate. Die paketorientierte Kommunikation wird im Socket-Interface auch als Datagram-Socket bezeichnet. Datagram-Sockets bieten die Möglichkeit, Pakete gleichzeitig an viele Empfänger zu verteilen. Sollen die Daten sicher übertragen werden, muss die Applikation dies mit einem geeigneten Protokoll sicherstellen.

Ethernet bietet in lokalen Netzwerken mit Hilfe von Multicast eine ideale Grundlage für die schnelle Datenverteilung auf viele Rechner in einem Cluster. Hierbei muss jedoch berücksichtigt werden, dass die Kommunikation über Multicast keine zuverlässige Datenübertragung darstellt.

2.4 Parallele Bildberechnung

Seit es möglich ist, auf einem Computer dreidimensionale Bilder zu berechnen, wurden Versuche unternommen, die Bildberechnung durch eine Parallelisierung zu beschleunigen. Dies ist naheliegend, da es technisch wesentlich einfacher ist, zwei CPUs zu verwenden, als eine einzige CPU um den Faktor zwei zu beschleunigen. Ein Großteil der parallelen Bildberechnungsverfahren wurde für die Implementierung in Hardware oder für Multiprozessorsysteme entworfen. Beispiele hierfür sind PixelFlow [MEP92], InfiniteReality [MBD97] oder Pomegranate [EIH00]. Heute werden parallele Bildberechnungsverfahren in allen Grafiksystemen eingesetzt. Selbst Grafikkarten, die für Computerspiele eingesetzt werden, beinhalten eine Vielzahl von parallel arbeitenden Funktionseinheiten. Spezialisierte Verfahren für den Einsatz auf vernetzten Workstations kamen erst mit der rapiden Leistungssteigerung der Grafikkarten für PCs Ende der 90er Jahre auf [AhP98,SFL00,NgZ00,SFL01].

Verfahren, die für den Einsatz in einem Cluster entwickelt wurden, versuchen, die Datenmenge entweder durch Kompression [AhP98] oder durch einen optimierten Nachrichtenaustausch [SFL00,SFK01] zu minimieren. Dies ist erforderlich, da die zur Verfügung stehende Übertragungsbandbreite in einem Cluster meist wesentlich geringer ist als in einer Hardware-Implementierung oder auf einem Rechner mit mehreren Prozessoren.

Um den Vergleich unterschiedlicher paralleler Bildberechnungsverfahren zu vereinfachen, wird in dieser Arbeit die Klassifikation von Steven Molnar [MCE94] übernommen. Der folgende Abschnitt gibt einen kurzen Überblick über diese Klassifikation.

2.4.1 Klassifikation von Algorithmen zur paralleler Bildberechnung

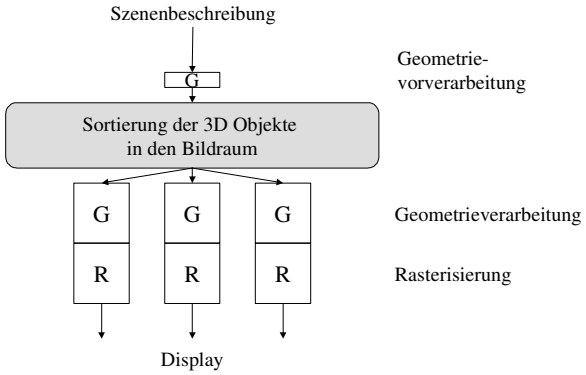

Abbildung 2-5: Sort-First

Parallele Bildberechnungsverfahren lassen sich in verschiedene Kategorien unterteilen. Weitgehend durchgesetzt hat sich die von Molnar, Cox, Ellsworth und Fuchs in [MCE94] vorgeschlagene Kategorisierung nach Sort-First, Sort-Middle und Sort-Last. Die Bildberechnung wird hierbei als Sortierung der grafischen Primitive in den Bildspeicher betrachtet. Damit diese Sortierung auch in parallelen Systemen funktioniert, müssen Informationen zwischen den einzelnen parallelen Instanzen ausgetauscht werden.

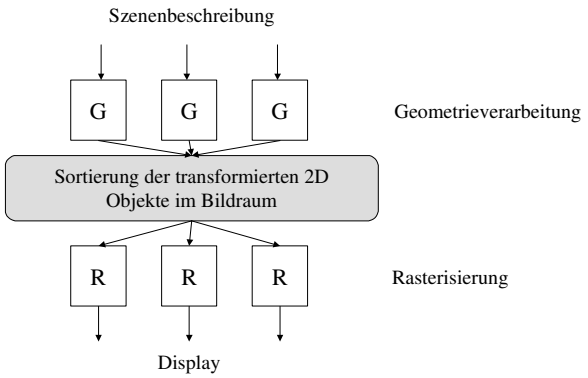

Abbildung 2-6: Sort-Middle

Betrachtet man eine vereinfachte Bildberechnungs-Pipeline, bestehend aus Geometrieverarbeitung und Rasterisierung, so kann die Sortierung an drei Stellen durchgeführt werden. Erfolgt die Sortierung zu Beginn der Geometrieverarbeitung, spricht man von einem Sort-First-Verfahren. Wird die Sortierung zwischen Geometrieverarbeitung und Rasterisierung durchgeführt, spricht man

von einem Sort-Middle-Verfahren. Bei einem Sort-Last-Verfahren erfolgt die Sortierung am Ende der Rasterisierung.

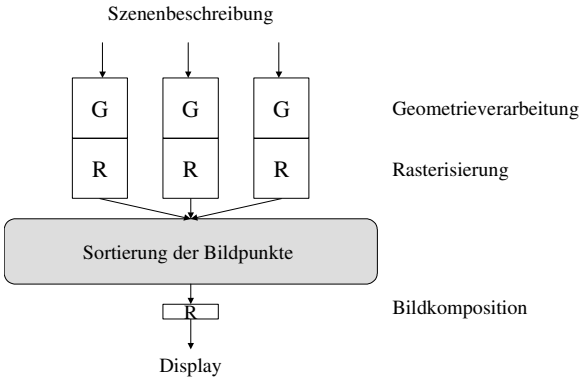

Abbildung 2-7: Sort-Last

Neben diesen drei Kategorien führen Crockett [Cro95] und Ellsworth [Ell96] noch die funktionale und die Frame- oder Zeit-Parallelität auf. Nicht jedes parallele Bildberechnungsverfahren lässt sich exakt einer Kategorie zuordnen. Um eine optimale Performanz zu erreichen, werden meist verschiedene Techniken kombiniert.

Die folgenden Abschnitte gehen auf die einzelnen Formen der Parallelität ein. Hierbei wird jedes Verfahren auf seine Eignung für die Bildberechnung in einem Cluster untersucht.

2.4.2 Sort-First

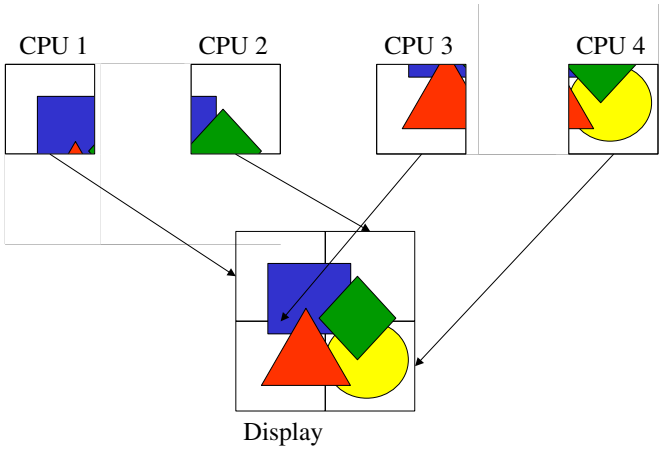

Abbildung 2-8: Sort-First / Bildbereichsparallelität

Der Bildbereich wird in Regionen unterteilt. Jeder Region wird eine Bildberechnungseinheit zugeordnet, die für die vollständige Berechnung dieser Region zuständig ist [CrO91,CKS02]. Der Bildbereich kann statisch als Gitter oder auch dynamisch nach der Komplexität einzelner Bildbereiche eingeteilt werden [Whi94, Mue95, Mue00]. Parallele Bildberechnungsverfahren auf der Basis von Sort-First werden in der Literatur häufig auch als "Screen space parallelism" bezeichnet.

Eine einfache Umsetzung von Sort-First teilt den Bildbereich in Bereiche gleicher Größe auf [CKS02]. Der Bildbereich kann in Spalten, Zeilen oder als Gitter unterteilt werden. Die gleichmäßige Lastverteilung hängt hierbei stark von der darzustellenden Szene ab. Eine bessere Lastverteilung lässt sich erreichen, wenn Form und Größe der Bildbereiche dynamisch an die Szene und die Position der virtuellen Kamera angepasst werden [Whi94,Mue95,Mue00]. Die dynamische Lastverteilung ist rechenintensiv und stellt einen potentiellen Flaschenhals bei der Parallelisierung dar. Daher wird bei der Verwendung von vielen Prozessoren häufig auf eine dynamische Lastverteilung verzichtet [MBD97].

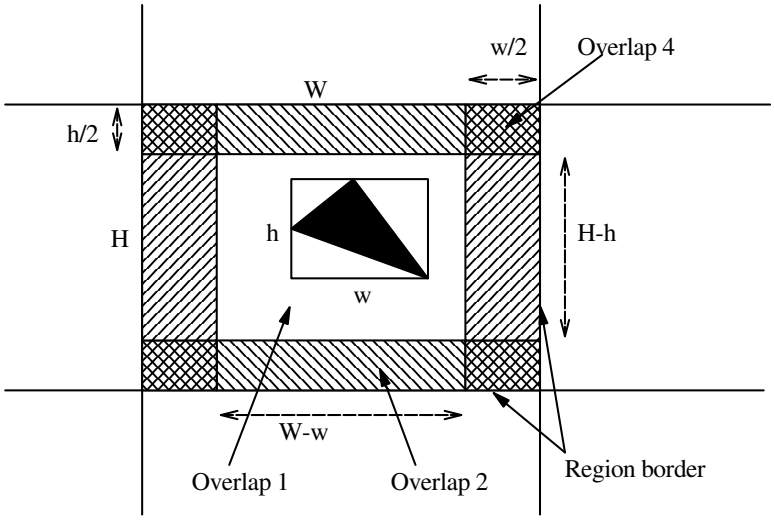

Abbildung 2-9: Überlappungsfaktor bei Sort-First

Wird der Bildschirm in Bereiche aufgeteilt, die parallel berechnet werden sollen, so müssen Polygone, die in mehr als einem Bereich sichtbar sind, mehrfach bearbeitet werden. Die Anzahl der mehrfach zu berechnenden Polygone steigt mit der Anzahl der Bereiche. Aus diesem Grund sind Sort-First-basierte parallele Bildberechnungsverfahren nur bedingt skalierbar.

Steven Molnar beschreibt in [MCE94] eine von John Eyles vorgeschlagene Abschätzung des Überlappungsfaktors. Dieser Faktor gibt an, wie häufig ein grafisches Primitiv durchschnittlich bearbeitet wird. In der Abschätzung von Eyles wird zur Vereinfachung ein grafisches Primitiv durch ein umschließendes Rechteck ersetzt. Wird der Bildschirm in Bereiche der Größe W x H eingeteilt, und ein grafisches Primitiv hat eine durchschnittliche Größe von w x h, so lässt sich der Überlappungsfaktor O mit folgender Formel abschätzen:

$$O = \frac{W + w}{W} \cdot \frac{H + h}{H}$$

Diese Formel lässt sich, wie in Abbildung 2-9 zu sehen, auch grafisch herleiten. Hierbei wird das Dreieck durch ein umschließendes Rechteck angenähert. Anhand des Mittelpunktes des Rechtecks lassen sich drei Fälle unterscheiden. Liegt der Mittelpunkt im inneren, nicht schraffierten Feld, so

ist das Dreieck nur in einem Bereich zu sehen. Der Überlappungsfaktor hierfür beträgt 1. Liegt der Mittelpunkt in den einfach schraffierten Bereichen, dann ist das Objekt auch im Nachbarbereich zu sehen. Fällt der Mittelpunkt in die Eckbereiche, dann ist das umschließende Rechteck gleichzeitig in vier Bereichen sichtbar. Der Überlappungsfaktor ist für diesen Fall 4. Bestimmt man nun die Wahrscheinlichkeiten, gewichtet mit den Überlappungsfaktoren, so erhält man den Ausdruck

$$O = \frac{(W-w)(H-h)+(H-h)w+(W-w)h+w\cdot h}{W\cdot H}$$

Fasst man die einzelnen Terme zusammen, so gelangt man zur oben beschriebenen Abschätzungsformel. Die von Molnar und Eyles entwickelte Formel ist auch gültig für W < w und H < h. Mit Hilfe dieser Formel lässt sich die potentielle Beschleunigung durch die Parallelisierung der Bildberechnung ermitteln.

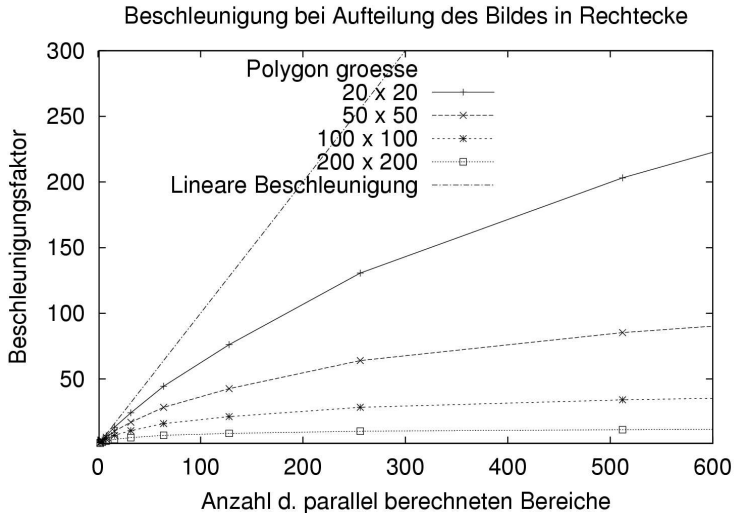

Abbildung 2-10: Skalierungsfaktor bei Sort-First für rechteckige Bereiche

Abbildung 2-10 und Abbildung 2-11 zeigen Simulationsergebnisse für verschiedene durchschnittliche Polygongrößen. Man sieht deutlich, dass die maximale Beschleunigung stark von der durchschnittlichen Polygongröße abhängig ist. Neben der Polygongröße spielt auch die Form der Bildschirmbereiche eine große Rolle.

Michael Cox und Narandra Bhandari zeigen in [CoB97], dass für ein Seitenverhältnis von 1 der geringste Überlappungsfaktor zu erwarten ist. In Abbildung 2-11 sind die Ergebnisse einer Simulation abzulesen, in der der Bildbereich in Streifen eingeteilt wird. Die erzielten Beschleunigungsfaktoren sind wesentlich geringer als bei der Aufteilung in Bereiche mit kleinem Seitenverhältnis.

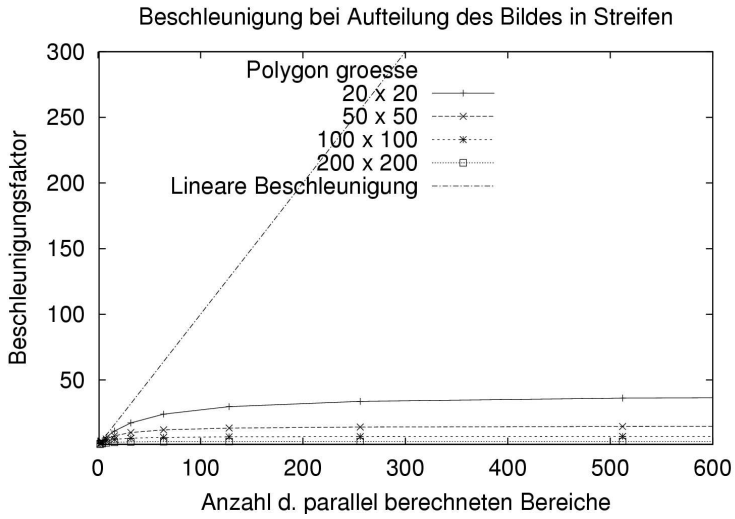

Abbildung 2-11: Skalierungsfaktor bei Sort-First und einer Aufteilung in Streifen

In [CSI98] untersucht Milten Chen den Einfluss des Überlappungsfaktors analytisch und anhand von Simulationsdaten. In dieser Untersuchung werden quadratische Bildbereiche verwendet. Milten Chen unterscheidet in seiner Arbeit Berechnungskosten für die Geometrieberechnung und Kosten für die Rasterisierung. Lediglich die Geometrieberechnung ist abhängig vom Überlappungsfaktor. Wenn die Berechnungszeit einer Szene hauptsächlich durch die Rasterisierung bestimmt ist, hat der Überlappungsfaktor einen geringeren Einfluss, und somit ist die erreichbare Beschleunigung größer als durch die Eyles/Molnar-Abschätzung vorhergesagt. Diese Tatsache macht Sort-First auch für komplexe Materialberechnungen mit Hilfen von Vertex-Shadern interessant.

Die parallele Bildberechnung auf der Basis von Sort-First ist sehr gut für den Einsatz in einem Cluster geeignet. Die erforderliche Datenrate ist konstant, da in jedem Frame maximal ein vollständiges Bild mit allen Farbwerten übertragen werden muss. Bei einer Auflösung von 1024 x 768 Bildpunkten müssen pro Frame etwa 2,5 MBytes übertragen werden. In einem Gigabit-Netzwerk lässt sich diese Datenmenge mit interaktiven Bildwiederholraten bewältigen.

2.4.3 Sort-Middle

Mehrere parallel arbeitende Prozessoren zur Geometrieverarbeitung berechnen grafische Primitive in Bildschirm-Koordinaten. Diese Primitive werden an mehrere parallel arbeitende Rasterisierungseinheiten verteilt. Es ist ein häufig verwendeter Ansatz für die Implementierung in Hardware. [MDB97,EIH00, CoB97]. Sort-Middle erfordert den schnellen Zugriff auf Ergebnisse der Geometrieeinheit. Dies lässt sich jedoch in Clustern aus Standardkomponenten nur schwer realisieren. Aus diesem Grund findet dieses Verfahren für die parallele Bildberechnung in Clustern keine Verwendung.

2.4.4 Sort-Last

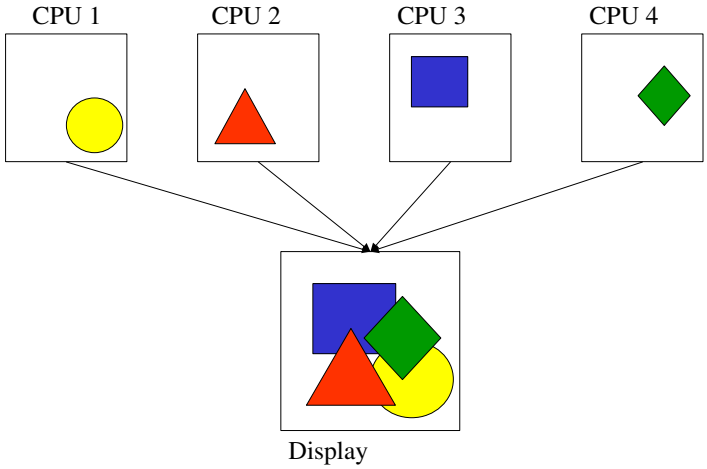

Abbildung 2-12: Sort-Last / Szenenparallelität

Teile der Szene werden auf parallel arbeitende Prozessoren verteilt. Jeder Prozessor berechnet ein Bild mit allen Teilen der Szene, die ihm zugeordnet wurden. Sind alle Einzelbilder berechnet, wird unter Berücksichtigung von Tiefeninformationen das vollständige Ergebnisbild zusammengefügt. Dieses Ergebnisbild beinhaltet wieder alle Teile der Szene [Mol91, MEP92, AhP98, NgZ00]. Die finale Bildkomposition erfordert einen sehr hohen Kommunikationsaufwand. Es wurden spezielle Bildkompositionsverfahren entwickelt, um diesen Aufwand zu minimieren [MJH94, MPH94, LRN96]. Die parallele Bildberechnung auf der Basis von Sort-Last wird in der Literatur häufig auch als "Objekt space parallelism“ bezeichnet.

Die Aufteilung der Szene ist meist statisch [Hei94]. Es ist jedoch auch möglich, nur die momentan sichtbaren Objekte einer Szene auf die parallel arbeitenden Prozessoren zu verteilen [SFL00]. Da die parallel arbeitenden Rechner nicht die komplette Szene im Hauptspeicher halten müssen, ist es möglich, in einem Cluster mit Hilfe von Sort-Last sehr große Modelle anzuzeigen. Die Berechnung der Teilszenen ist unabhängig voneinander. Die Bildberechnung lässt sich theoretisch beliebig beschleunigen. Sort-Last wird meist zur Darstellung von Volumendaten eingesetzt [Neu93,Mph94,Law96,YCC99]. Das Verfahren kann jedoch auch zur Darstellung polygonaler Modelle verwendet werden [KAO98].

Bei allen parallelen Algorithmen zur Bildberechnung, die auf dem Sort-Last-Prinzip basieren, werden mehrere Bilder berechnet, die jeweils nur einen Teil der Szene darstellen. Aus diesen einzelnen Bildern muss das Ergebnisbild berechnet werden, so dass alle Teile der Szene in einem einzigen Bild enthalten sind. Die Einzelbilder können Objekte in beliebigen Entfernungen zum Betrachter darstellen. Aus diesem Grund sind für die korrekte Zusammenfügung des Ergebnisbildes Informationen über die Tiefe aller Bildpunkte erforderlich. In das Ergebnisbild wird für jeden Bildpunkt der Farbwert aus den einzelnen Bildern übernommen, der dem Betrachter am nächsten liegt.

Die Entfernung kann für jeden Bildpunkt aus dem Z-Buffer entnommen werden, der beim Zeichnen der Teilszenen gefüllt wird. Wurde die Szene in räumlich nicht überlappende Bereiche aufgeteilt, kann bei der Bildkomposition auf eine Untersuchung der Entfernung jedes einzelnen Bildpunktes verzichtet werden. Das Ergebnisbild wird in diesem Fall durch Überlagerung der Einzelbilder von hinten nach vorne erzeugt [NgZ00].

Für die Bild-Komposition müssen sehr große Datenmengen bewältigt werden. Am Beispiel von 32 Prozessoren, die gemeinsam ein Bild mit einer Auflösung von 1024 x 768 Bildpunkten berechnen, soll dies verdeutlicht werden. Bei einer Standard-Farbtiefe von 24 Bit und einer Z-Buffer-Genauigkeit von 32 Bit werden für jeden Bildpunkt 7 Bytes benötigt. Für ein komplettes Bild müssen etwa 5,5 MByte übertragen werden. Soll nun aus den 32 einzelnen Bildern ein Ergebnisbild berechnet werden, so müssen die Daten von mindestens 31 Bildern zwischen den Prozessoren ausgetauscht werden. Für ein Ergebnisbild müssen also etwa 170 MByte übertragen werden. Sollen nun Bilder in interaktiven Bildwiederholraten berechnet werden, muss die Kommunikations-Infrastruktur ein Datenvolumen von etwa 4 GByte/s bewältigen. Eine Möglichkeit, die Datenmenge zu reduzieren, besteht darin, nur die Bildpunkte zur Komposition zu versenden, die von einem Objekt verdeckt werden. Meist wird für die Bestimmung aktiver Punkte ein umschließendes Rechteck um einzelne Polygone oder um Gruppen von Polygonen verwendet. Wenn nicht alle Bildpunkte für die Bild-Komposition verwendet werden, spricht man auch von einem Sort-Last-Sparse-Verfahren im Gegensatz zu einem Sort-Last-Full-Verfahren [MCE94,YCC99]. Der Erfolg dieser Strategie hängt sehr stark von der Verteilung der sichtbaren Objekte im Bildraum ab.

Das Datenvolumen lässt sich auch durch Kompression reduzieren. Häufig werden hierfür einfache und schnelle Algorithmen eingesetzt. Beispielsweise wird in [LaL94] und [AhP98] eine Run-Length-Kodierung verwendet, um die Bilder verlustfrei zu komprimieren. Es muss jedoch ein Kompromiss zwischen Kompressionsrate und Kompressionsaufwand gefunden werden. Eine Kompression ist nur dann sinnvoll, wenn sie den dafür erforderlichen Rechenaufwand durch Einsparungen bei der Datenübertragungszeit aufwiegt.

Die sehr großen Datenmengen, die zur Bildkomposition versendet werden müssen, können auch durch eine Kompression nicht in dem Maße verringert werden, dass die Datenübertragung mit geringem Hardware-Aufwand zu bewältigen ist. Um dies zu erreichen, wurden verschiedene Verfahren entwickelt, die den Arbeitsaufwand auf möglichst viele parallel arbeitende Prozessoren und parallel betriebene Kommunikationswege verteilen. Weit verbreitete Verfahren sind Binary Swap [MPH94] und Direkt Send [Neu93,MWP01]. Bei diesen Verfahren wird die Bildkomposition nicht von einem einzelnen Prozessor durchgeführt, sondern parallel von allen beteiligten Rechnern.

Ein weiteres Problem bei der Verwendung von Sort-Last besteht darin, dass jeder Rechner Teilbilder in der vollen Auflösung des Displays liefern muss. Die Auflösung von gekachelten Projektionen kann jedoch schnell die maximale Auflösung der Grafik-Hardware überschreiten. In [MWP01] wird dieses Problem näher erläutert. Bei der vorgeschlagenen Lösung werden die

hochauflösenden Teilbilder in kleinen Teilen berechnet. Dies entspricht jedoch einem Sort-First-Ansatz, und es treten daher ähnliche Skalierungsprobleme auf.

2.4.5 Funktionale Parallelität

Bei der funktionalen Parallelität werden einzelne Teilaufgaben von spezialisierten Funktionseinheiten parallel ausgeführt. Häufig werden hierbei die Funktionseinheiten in der Form einer Pipeline angeordnet. Ein mögliches Anwendungsszenario ist die OpenGL-Pipeline. Während die Bildpunkte eines Polygons berechnet werden, können bereits die Eckpunkte des nächsten Polygons in den Bildraum transformiert werden. Transformations- und Rasterisierungseinheit können parallel betrieben werden. Mit Hilfe der funktionalen Parallelität lässt sich eine einzelne Befehlsfolge auf einfache Weise parallelisieren. Allerdings lässt sich ein einzelner Befehl oder eine einzelne Grafikoperation nicht in beliebig viele Teilaufgaben zerlegen. Die Performanz einer Pipeline wird durch die Geschwindigkeit der langsamsten Funktionseinheit bestimmt.

2.4.6 Frame-Parallelität

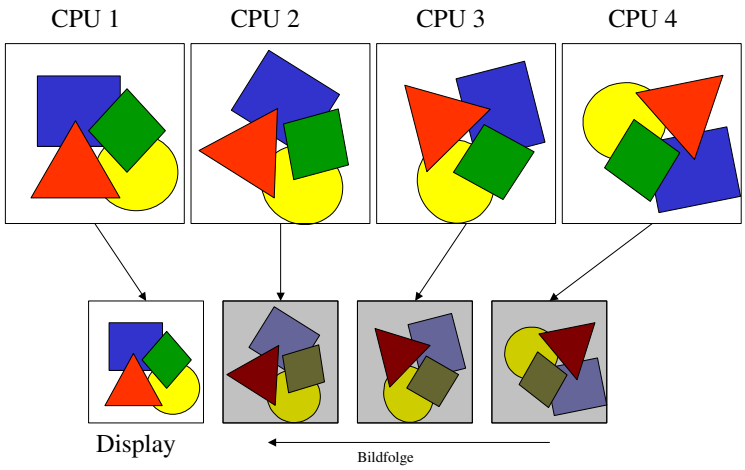

Abbildung 2-13: Frame-Parallelität

Jedes Frame einer Animationssequenz wird von einem einzelnen Prozessor berechnet. Die Berechnungszeit für ein einzelnes Bild wird hierbei nicht beschleunigt. Erhöht man die Anzahl der Prozessoren, so werden mehr Bilder der Animationssequenz pro Zeiteinheit fertiggestellt. Mit Hilfe der Frame-Parallelität lässt sich auf einfache Weise die Bildwiederholrate erhöhen. Theoretisch lässt sich das Verfahren beliebig skalieren. Benötigt beispielsweise die Berechnung eines einzelnen Bildes zehn Sekunden, so können mit 100 Prozessoren zehn Bilder pro Sekunde angezeigt werden. Da die Berechnungszeit für ein einzelnes Bild jedoch nicht beschleunigt wird, bleibt eine hohe Verzögerungszeit zwischen einer Benutzereingabe und der entsprechenden Auswirkung auf die Darstellung bestehen. Im obigen Beispiel werden zwar zehn Bilder pro Sekunde angezeigt, es dauert jedoch zehn Sekunden, bis sich eine Benutzereingabe auf die Anzeige auswirkt. Das Verfahren eignet sich daher nur für solche Anwendungen, bei denen die nicht parallele

Bildberechnung bereits Interaktionen ermöglicht. In einem solchen Fall kann die Bildwiederholrate angehoben werden, um eine flüssige Anzeige zu erreichen.

2.4.7 Hybride Verfahren

Bei der parallelen Bildberechnung werden meist verschiedene Techniken kombiniert, um eine optimale Performanz zu erreichen. Ein Beispiel hierfür ist das hybride Sort-First- und Sort-Last-Verfahren von Rudrajit Samanta u.a. [SFL00]. Das Verfahren basiert primär auf dem Sort-Last-Prinzip. Es wird jedoch nicht nur die Szene auf die parallel arbeitenden Rechner verteilt, sondern bei der Verteilung wird auch darauf geachtet, dass die zugeteilten Objekte im Bildbereich nahe beieinander liegen. In einem Bildbereich, in dem nur Objekte eines einzelnen Rechners sichtbar sind, ist keine aufwendige Bildkomposition mit Hilfe des Z-Buffers erforderlich. Auf diese Weise lässt sich die Datenmenge im Vergleich zu einem reinen Sort-Last-Verfahren wesentlich reduzieren. Die Effizienz des Verfahrens hängt im Wesentlichen von der Verteilung der dargestellten Objekte im Bildraum ab. Mit steigender Anzahl parallel berechneter Bereiche steigt auch meist der Anteil der Bildfläche, für die Sort-Last eingesetzt werden muss.

Bei der parallelen Bildberechnung werden meist Einzelbilder als Zwischenergebnis berechnet, die anschließend in einem weiteren Verarbeitungsschritt zu einem Ergebnisbild zusammengefasst werden müssen. Die Datenübertragung kann entweder über ein Standard-Netzwerk erfolgen oder mit Hilfe spezieller Hardware durchgeführt werden. Der folgende Abschnitt gibt einen Überblick über Hardwarekomponenten zur Beschleunigung der Bildkomposition in einem Cluster.

2.5 Hardware für die Bildkomposition in einem Cluster

Die Performanz bei der parallelen Bildberechnung in einem Cluster wird meist von zwei Faktoren bestimmt. Dies sind die Netzwerkbandbreite und die Bandbreite der Datenübertragung zwischen Hauptspeicher und Grafikspeicher. Beide Faktoren sind beim Einsatz von Standardkomponenten kaum zu beeinflussen. Bei der Umsetzung eines parallelen Verfahrens kann eine vollständige Umsetzung in einer hochspezialisierten Hardware erfolgen, oder es kann versucht werden, eine Umsetzung mit Standardkomponenten zu erreichen. Der Spezialhardware-Ansatz ermöglicht eine sehr hohe Performanz, er erfordert jedoch einen erheblichen Investitionsaufwand. Demgegenüber ist der Einsatz von Standardkomponenten preisgünstig, es wird jedoch eine geringere Performanz erreicht. Zur Lösung dieses Dilemmas muss ein Kompromiss zwischen Standard- und Spezialhardware gefunden werden. Im Laufe der letzten Jahre wurden einige Systeme entwickelt, die darauf abzielen, das Übertragen und Zusammenfügen von Bildteilen zu beschleunigen. Mit Hilfe dieser Zusatzhardware kann die parallele Bildberechnung auf vernetzten Workstations wesentlich beschleunigt und das Verbindungsnetzwerk entlastet werden.

Abbildung 2-14: Scalable Graphics Engine von IBM

2.5.1 Scalable Graphics Engine von IBM

IBM stellt in [PeJ01] eine skalierbare Grafik-Hardware zur Ansteuerung hochauflösender Displays vor. Die so genannte *Scalable Graphics Engine 3* (SGE3) besitzt mehrere Grafikausgänge und wird über mehrere Ethernet-Anschlüsse mit einem Cluster verbunden. Die einzelnen Rechner des Clusters lesen den Inhalt ihres lokalen Bildspeichers aus und senden ihn an die SGE3. Dort können die eintreffenden Daten miteinander kombiniert und auf verschiedene Grafikausgänge verteilt werden. Ein Problem dieses Ansatzes besteht in der relativ langsamen Geschwindigkeit, mit der Bilddaten von der Grafikkarte in den Hauptspeicher und anschließend über das Netzwerk übertragen werden. Die SGE3 ist sicher gut für den Einsatz in Sort-First-basierten Verfahren geeignet. Der Versuch, einen Sort-Last-basierten Ansatz mit Hilfe der SGE3 zu realisieren, dürfte jedoch an der zu geringen Datenübertragungsgeschwindigkeit aktueller Grafikkarten scheitern. Die vorgestellte Hardware hat den großen Vorteil, dass sie sich ohne Modifikationen an der bestehenden Hardware in ein Cluster-System integrieren lässt.

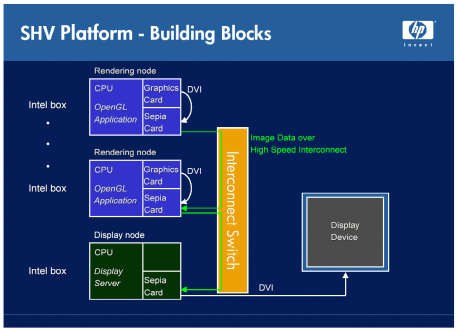

Abbildung 2-15: Sepia (High Performance Technical Computing Division of HP)

2.5.2 HP Sepia

Sepia wurde von Compaq entwickelt und 1999 von Laurent Moll u.a. in [MHS99] vorgestellt. Sepia ist eine Hardware, die speziell für die Bildkomposition in einem Cluster entwickelt wurde. Die Operationen, die bei der Bildkomposition durchgeführt werden, lassen sich sehr flexibel konfigurieren. Dadurch ist es möglich, sehr unterschiedliche Kompositionsverfahren zu realisieren. Sepia ist als PC-Erweiterungskarte realisiert. Mehrere Rechner werden in einer Reihe als Pipeline zusammengeschaltet. Die Bilddaten werden in dieser Pipeline von Rechner zu Rechner

weitergereicht. Jeder Rechner führt mit den ankommenden und den lokal gespeicherten Daten die konfigurierten Bildoperationen durch. Die Bilddaten müssen für die Bildkomposition aus dem Bildspeicher ausgelesen und an die Kompositionseinheit übergeben werden. Hierbei ist Sepia an die begrenzte Übertragungskapazität der Grafikkarte gebunden. In [MHS99] wird für spätere Entwicklungen ein direkter Zugriff auf den Speicher der Grafikkarte eingeplant. Bis heute ist jedoch keine Grafikkarte verfügbar, die diese Funktion bereitstellt. Im Gegensatz zur 1999 präsentierten Version von Sepia wurde im Laufe der weiteren Entwicklung das direkte Speicherinterface zur Grafikkarte verworfen [HP04]. Heute unterstützt Sepia die Übertragung der Bilddaten über das DVI-Interface. Das Verbindungsnetzwerk wird im aktuellen Entwurf mit Infiniband 4x realisiert. Im Gegensatz zum ursprünglichen Entwurf, ist neben der Konfiguration als Pipeline auch eine Konfiguration über einen Netzwerk-Switch vorgesehen.

2.5.3 Lightning-2

Lightning-2 [SEP01] ist ein externer Bildspeicher, mit dem die Rechner eines Clusters verbunden werden. Im Gegensatz zur frühen Sepia-Implementierung oder SGE3 von IBM werden die Rechner über den digitalen Ausgang der Grafikkarte (DVI) an den Bildspeicher angebunden. Der digitale Ausgang einer Grafikkarte wird normalerweise direkt zum Ansteuern eines Bildschirms verwendet. Über diesen Ausgang können mehr als 60 hochaufgelöste Bilder pro Sekunde verlustfrei gesendet werden. Aufgrund der sehr hohen Verbreitung von DVI lässt sich die Anbindung des Bildspeichers günstig realisieren. Für das Senden eines Bildes an die Kompositionseinheit müssen die Bilddaten nicht mehr in den Hauptspeicher kopiert werden, sondern sie werden direkt über das Video-Interface gesendet. Über den digitalen Videoausgang werden üblicherweise nur Farbwerte gesendet. Um eine parallele Bildberechnung auf der Basis von Sort-Last zu ermöglichen, müssen auch Tiefeninformationen übertragen werden. In Lightning-2 werden hierfür die Daten des Z-Buffers in den Farbpuffer übertragen. Einzelne Bildpunkte in der Grafikausgabe werden verwendet, um Lightning-2 die Bedeutung der aktuellen Ausgabe mitzuteilen.

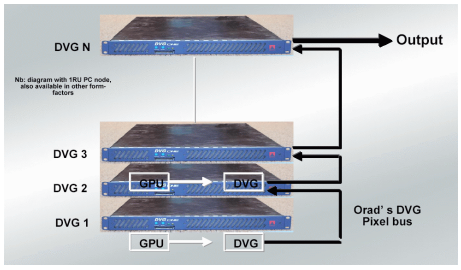

Abbildung 2-16: Orad DVG

2.5.4 ORAD DVG

Die Firma Orad stellt Systemkomponenten für die Videobearbeitung her. Mit dem DVG-System wurde eine Cluster-Hardware entwickelt, um den Einsatz mehrerer PCs zur Darstellung hochwertiger Videoströme zu ermöglichen. Das System besteht aus mehreren PCs, die um eine Kompositionshardware erweitert werden. Die Bilddaten werden direkt über den digitalen Videoausgang ausgelesen. Für die eigentliche Bildkomposition stehen verschiedene Operationen

zur Verfügung. Das Haupteinsatzgebiet liegt in der Erstellung qualitativ hochwertiger Bilder mit komplexen Verfahren zur Kantenglättung. Die Hardware kann jedoch auch zur Bildkomposition bei Sort-First-basierten Verfahren eingesetzt werden.

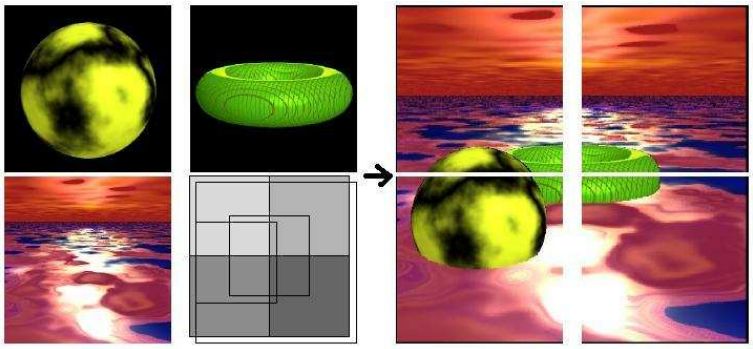

Abbildung 2-17: Metabuffer: Bildkomposition mit Z und Alpha. (Quelle [BBF00])

2.5.5 Metabuffer

Der *Metabuffer* [BBF00] ist ein hardware-basierter Grafikspeicher, an den mehrere PCs über den digitalen Videoausgang (DVI) angeschlossen werden. Es werden RGB-Farbwerte, ein Alpha-Kanal und der Z-Wert an den Metabuffer übergeben. Um den Z-Wert ohne Manipulation der Grafikhardware über das digitale Videointerface übertragen zu können, werden im Speicher der Grafikkarte abwechselnd der RGBA-Wert und der Z-Wert angezeigt. Die Hardware kombiniert alle ankommenden Bilder zu einem Ergebnisbild, das an einem oder mehreren Ausgängen ausgegeben wird. Die Metabuffer-Hardware ist zusätzlich in der Lage, ankommende Bildinformationen zu skalieren. Dadurch lassen sich einzelne Bildteile mit unterschiedlicher Auflösung darstellen. Bei der in [BBF00] vorgestellten Hardware handelt es sich jedoch nicht um ein am Markt erhältliches Produkt. Die Funktionsfähigkeit des Konzeptes wurde zum großen Teil mit einer Software-Simulation überprüft. In [ZBB01] wird die Verwendung des Metabuffers zur Darstellung von ISO-Flächen aus sehr großen Volumendatensätzen beschrieben.

2.5.6 Zukünftige Hardwareentwicklungen

Ein allgemeiner Trend hin zu einer hardwarebeschleunigten parallelen Bildberechnung in Clustern ist noch nicht zu erkennen. Dies liegt einerseits daran, dass die am Markt erhältliche Hardware erst einen sehr frühen Entwicklungsstand erreicht hat und andererseits daran, dass viele Firmen den Umstieg auf Cluster-Systeme aufgrund der hohen Umstellungskosten für vorhandene Software scheuen. Dies wird sich erst mit der Verfügbarkeit von leistungsfähigen Softwaresystemen zur Bildberechnung in Clustern ändern.

In Zukunft wird es eine engere Zusammenarbeit zwischen Herstellern von Grafikchips für den Massenmarkt und Herstellern von Hochleistungsrechen geben. Eine solche Kooperation sind beispielsweise SGI und ATI eingegangen. Aus dieser Zusammenarbeit werden Anforderungen, die sich aus der parallelen Bildberechnung ergeben, in das Design zukünftiger 3D-Chips einfließen. Beispielsweise spielt der schnelle Zugriff auf den Grafikkartenspeicher für Computerspiele kaum

eine Rolle. Diese Operation ist jedoch die Voraussetzung für eine schnelle Bildkomposition bei der parallelen Bildberechnung. Hardware zur Bildkomposition kann aus günstigen Standardkomponenten hergestellt werden. Wenn eine breite Unterstützung von Softwaresystemen gewährleistet ist, wird die hardwarebeschleunigte Bildkomposition diejenige per Software ablösen.

2.6 Szenengraphen

Ein Szenengraph ist eine Datenstruktur, mit dessen Hilfe eine dreidimensionale Szene beschrieben wird. Der Szenengraph enthält alle Informationen, die für die Erzeugung einer dreidimensionalen Darstellung benötigt werden. Dies umfasst Materialeigenschaften, Lichtquellen und die geometrische Beschreibung der Szene. Ein Szenengraph wird in der Regel in Zusammenhang mit einem Szenengraphensystem verwendet. Das Szenengraphensystem bietet eine Schnittstelle zur Durchführung von Operationen auf dem Graphen. Die wichtigste Operation ist die Berechnung einer dreidimensionalen Darstellung der Szene. Im Laufe der 80er und 90er Jahre wurden einige Szenengraphensysteme für VR-Anwendungen entwickelt. Eine weitere Verbreitung erreichten Open Inventor und OpenGL Performer von SGI. Da ein Wechsel des Szenengraphensystems sehr große Auswirkungen auf vorhandene Software hat, führt ein Umstieg auf ein anderes System zwangsläufig zu einem hohen Entwicklungsaufwand. Aus diesem Grund werden Inventor und Performer auch heute noch häufig eingesetzt. Zum Zeitpunkt der Entwicklung dieser beiden Szenengraphensysteme spielten vernetzte Workstations noch keine Rolle. Inventor und Performer sind daher für den Einsatz in Multiprozessorsystemen mit mehreren Grafikpipelines entwickelt und optimiert worden.

Es existiert bisher kein Szenengraphensystem, das direkt die Bildberechnung in einem Cluster-System unterstützt. Um mit Performer oder Inventor auf vernetzten Workstations dreidimensionale Bilder parallel zu berechnen, ist ein erheblicher Entwicklungsaufwand erforderlich [HSF99]. Alle Rechner, die an der Bildberechnung beteiligt sind, müssen auf einer gemeinsamen Datenbasis arbeiten. Werden Teile der Szene gelöscht, verändert oder neu geladen, so muss dies von allen parallel arbeitenden Rechnern berücksichtigt werden. Dies erfordert insbesondere in interaktiven Anwendungen einen erheblichen Kommunikationsaufwand.

2.7 Zusammenfassung

Damit ein Cluster aus vernetzten Workstations für die parallele Bildberechnung in interaktiven VR-/AR-Anwendungen eingesetzt werden kann, müssen effiziente Strategien für die Datenverteilung gefunden, parallele Bildberechnungs-Algorithmen angepasst und unterschiedliche Hardware eingebunden werden. Es ist damit zu rechnen, dass auch in Zukunft weitere verbesserte Verfahren und schnellere Hardware entwickelt werden. Beim Einsatz momentan verfügbarer Szenengraphensysteme muss die Bildberechnung im Cluster vollständig auf der Anwendungsebene implementiert werden. Dies stellt einen erheblichen Entwicklungsaufwand dar. Dieser lässt sich wesentlich verringern, wenn alle erforderlichen Komponenten in das Szenengraphensystem integriert werden. Ziel dieser Arbeit ist es, ein solches System bereitzustellen.

Im Folgenden wird der Entwurf und die Umsetzung eines Szenengraphensystems beschrieben, das neben Einzel- und Multiprozessorsystemen auch Cluster-Systeme unterstützt. Hierbei werden bestehende Techniken erweitert und neue Verfahren entwickelt. Da die Bildberechnung auf vernetzten Workstations ein aktives Forschungsfeld ist, wird beim Entwurf des Systems besonderer Wert auf Flexibilität und Erweiterbarkeit gelegt.

3 Entwurf eines verteilten Szenengraphensystems

In diesem Kapitel wird der Entwurf eines Systems für die parallele Bildberechnung in einem Cluster von Workstations vorgestellt. Die zentrale Komponente dieses Entwurfs ist der Szenengraph. Ausgehend von einem Szenengraphensystem für einen einzelnen Rechner werden Komponenten identifiziert, die für eine verteilte parallele Bildberechnung erforderlich sind. Ziel dieses Kapitels ist es, einen Überblick über die Gesamtarchitektur zu geben und anhand von Anwendungsszenarien die Einsatzmöglichkeiten des Gesamtsystems aufzuzeigen.

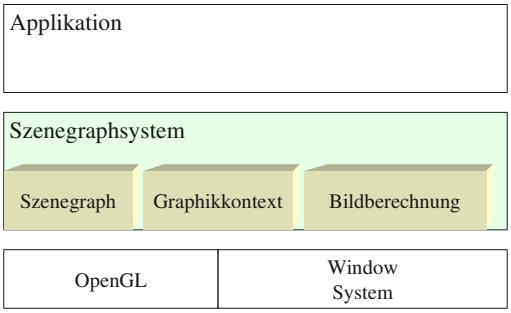

Abbildung 3-1: Lokale Bildberechnungspipeline

Der Einsatz eines Systems zur Verwaltung und Anzeige eines Szenengraphen ist heute in Applikationen aus den Bereichen VR und AR sehr weit verbreitet. Der Szenengraph enthält hierbei eine vollständige Beschreibung der darzustellenden Szene. Die eigentliche Bildberechnung wird nicht von der Applikation, sondern vom Szenengraphensystem durchgeführt. Neben der Beschreibung der Szene benötigt das Szenengraphensystem zusätzliche Informationen darüber, an welcher Stelle des Bildschirms und mit welchen Parametern die Bildberechnung erfolgen soll. Diese Informationen sind in der Regel nicht Teil des Szenengraphen, sondern sie werden als Kontext, in dem die Bildberechnung erfolgen soll, dem Szenengraphensystem übergeben. Abbildung 3-1 zeigt sehr vereinfacht die Funktionsweise eines Szenengraphensystems auf einem einzelnen Rechner. Der Szenengraph wird mit Hilfe des Grafikkontexts durch die Bildberechnung in den Bildspeicher der Grafikhardware projiziert. Die Anzeige erfolgt über einen Bildschirm oder über ein anderes Projektionssystem. Die Applikation ist vollständig von der Bildberechnung und somit auch von der eingesetzten Hardware getrennt.

3.1 Ein verteilter Szenengraph

Häufig eingesetzte Szenengraphensysteme wie z.B. die schon recht betagten Systeme Performer oder Inventor, aber auch neuere Entwicklungen wie Open Scenegraph (nicht zu verwechseln mit dem in dieser Arbeit verwendeten Szenengraphen OpenSG), bieten keine Unterstützung für die verteilte Bildberechnung in einem Cluster. Zum Zeitpunkt der Entwicklung von Performer und Inventor wurden Cluster von Workstations noch nicht für die parallele Bildberechnung eingesetzt. Vielmehr standen Multiprozessorsysteme mit vielen CPUs und mehreren Grafikpipelines im

Vordergrund. Auch bei neueren Entwicklungen wie z.B. Open Szenengraph wurde auf eine direkte Unterstützung von Clustern verzichtet. Diese fehlende Unterstützung bedeutet nicht, dass diese Systeme heute nicht in einer Cluster-Umgebung genutzt werden können. Es bedeutet jedoch, dass nicht das Szenengraphensystem, sondern die Applikation dafür sorgen muss, dass der Szenengraph auf verschiedene Rechner verteilt und parallel berechnet wird. Dies hat zur Folge, dass die oben beschriebene Trennung der Applikation von der eingesetzten Hardware nicht mehr aufrechterhalten wird. Eine mögliche Lösung hierfür besteht darin, zwischen Applikation und Szenengraphensystem eine Verteilungsebene einzuführen. In [HSF99] wird eine Verteilungsebene für Inventor beschrieben. Die beschriebene Verteilung wird in dieser Arbeit nicht für die parallele Bildberechnung, sondern für ein Mehrbenutzersystem eingesetzt. Allerdings müssen sowohl bei der parallelen Bildberechnung als auch bei einem Mehrbenutzersystem Änderungen des Szenengraphen über ein Netzwerk verteilt werden. Ein ähnlicher Ansatz zur Verteilung eines Szenengraphen wird in [ZHC00] beschrieben. Hier steht die Synchronisation verschiedener Szenengraphensysteme im Mittelpunkt. Aufbauend auf diesen Ansätzen ließe sich ein Cluster-Layer für lokale Szenengraphen entwickeln. Das Problem hierbei besteht jedoch darin, dass alle Änderungen des Szenengraphen registriert werden müssen. Dies ist häufig nur durch einen zeitaufwendigen Vergleich der Zustände des Szenengraphen möglich.

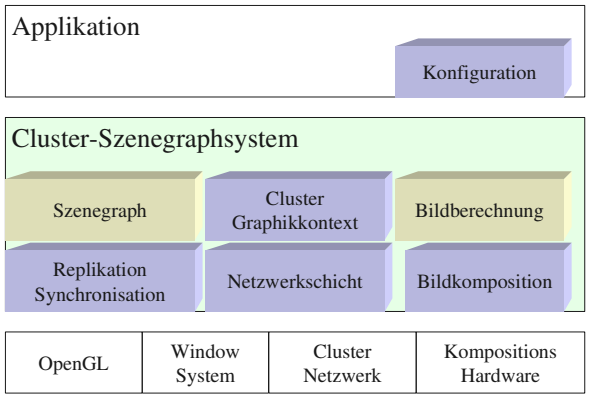

Abbildung 3-2: Komponenten eines Cluster-Szenengraphensystems

Die Entwicklungen der letzten Jahre haben gezeigt, dass die parallele Bildberechnung in Clustern ein sehr aktives Forschungsgebiet ist. Es wurden neue Verfahren und leistungsfähige Hardware für die parallelen Bildberechnung in einem Cluster entwickelt. Eine Applikation, die für die Datenverteilung und die parallele Bildberechnung verantwortlich ist, muss ständig an neue Entwicklungen angepasst werden. Dieses Problem soll in dem hier vorgestellten Entwurf durch eine strikte Trennung der Applikation von der eigentlichen Bildberechnung gelöst werden. Die Verwendung des Clusters soll vollständig von der Applikation getrennt werden.

Mit OpenSG wurde im Jahre 2000 die komplette Neuentwicklung eines Szenengraphen begonnen. Zu diesem Zeitpunkt war die steigende Bedeutung von Clustern bereits abzusehen. Grafikkarten für PCs erreichten zwar noch nicht die Leistung von speziellen Grafik-Rechnern wie beispielsweise die SGI Infinit Reality, aber sehr schnelle Produktzyklen und ein großer Massenmarkt ließen das Potential von Clustern auf der Basis von Standard-PC-Hardware bereits erahnen. Dies war der

Grund, warum die Unterstützung von Clustern von Beginn an in das Design von OpenSG eingeflossen ist. Der Cluster-Support von OpenSG ist das Ergebnis dieser Arbeit.

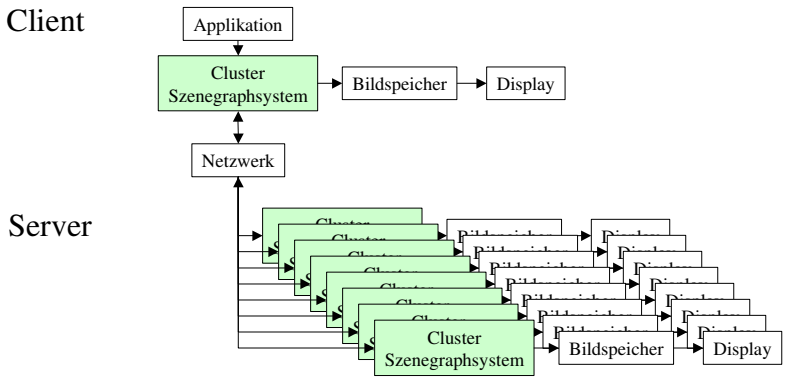

Abbildung 3-3: Parallele Bildberechnung in einem Cluster

Der hier vorgestellte Entwurf geht davon aus, dass die Applikation den Szenengraph und den Grafikkontext auf einem Rechner dem Szenengraphensystem übergibt. Das Szenengraphensystem verteilt den Szenengraph und den Grafikkontext auf alle Rechner im Cluster. Auf jedem Rechner im Cluster läuft eine Instanz des Szenengraphensystems. Abbildung 3-3 zeigt die in dieser Arbeit verwendete Nomenklatur für Client und Server. Als Client wird der Rechner bezeichnet, auf dem die Applikation ausgeführt wird. Alle anderen Rechner, die ihre Rechenleistung zur Verfügung stellen, werden als Server bezeichnet. Mit diesem Ansatz können beliebige parallele Bildberechnungen durchgeführt werden. Jeder Rechner kann sowohl als Ausgabe für ein Projektionssystem dienen als auch die Bildberechnung für einen anderen Rechner übernehmen.

Damit die Funktionen eines Szenengraphensystems in einem Netzwerk verteilt werden können, müssen die in Abbildung 3-1 aufgeführten Funktionsblöcke bereitgestellt werden. Um diese Erweiterung des Szenengraphenkonzeptes für die Applikation transparent zu halten, bietet sich als Schnittstelle der Grafikkontext an. In einem lokalen System wird durch ihn festgelegt, wie die Ausgabe der dreidimensionalen Daten auf den Bildschirm erfolgen soll. In einer verteilten Cluster-Umgebung wird der lokale Grafikkontext durch einen verteilten Grafikkontext ausgetauscht. Der verteilte Grafikkontext enthält zusätzliche Informationen über die beteiligten Rechner im Cluster und die Art der parallelen Bildberechnung.

Abbildung 3-4 zeigt stark vereinfacht die Komponenten eines verteilten Szenengraphen. Aus Sicht der Applikation sieht das Szenengraphensystem so aus, als wäre nur ein einziger Rechner an der Bildberechnung beteiligt. Alle Aufgaben der Kommunikation, Synchronisation und Replikation sowie der Übertragung und Zusammenfügung von Teilergebnissen der parallelen Bildberechnung sind für die Applikation nicht sichtbar. Im Folgenden werden die einzelnen Funktionsblöcke kurz beschrieben. Komplexere Funktionszusammenhänge werden in eigenen Kapiteln beschrieben.

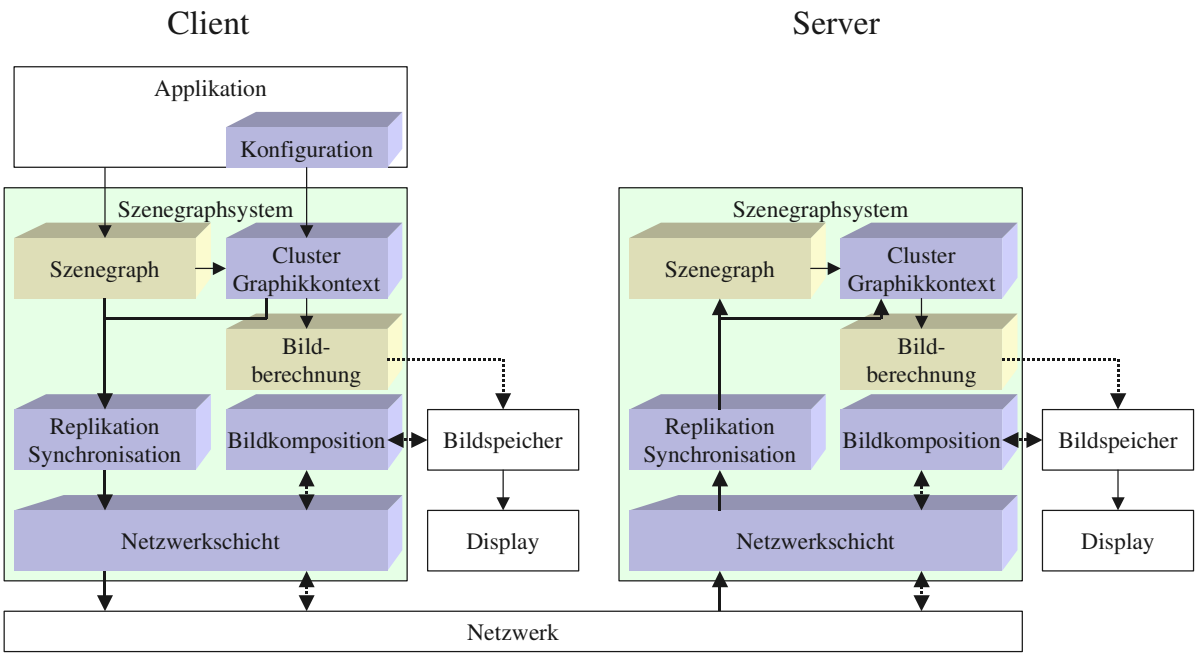

Abbildung 3-4: Datenfluss in einem verteilten Szenengraphen

- Cluster Grafikkontext: Der Cluster Grafikkontext bündelt alle Informationen, die für die parallele Bildberechnung erforderlich sind. Es wird festgelegt, welcher Rechner eine Projektion betreibt, welche Bildteile über ein Netzwerk übertragen werden müssen und wie die Lastverteilung innerhalb des Clusters durchgeführt wird.

- Netzwerkschicht: Der Szenengraph, Bilddaten und andere Informationen, die für die parallele Bildberechnung erforderlich sind, müssen über ein Netzwerk übertragen werden. Diese Netzwerkschicht muss die Daten effizient verteilen und unterschiedliche Netzwerkhardware wie beispielsweise Ethernet, Infiniband [Inf02] oder Myrinet [BCF95] unterstützen. Bei der Verteilung ist es wichtig, Protokolle zu nutzen, die eine schnelle Datenverteilung in einem Cluster ermöglichen.

- Bildkomposition: Werden einzelne Bildteile oder einzelne Teile der Szene parallel auf verschiedenen Rechnern berechnet, müssen die dabei entstandenen Teilbilder zu einem vollständigen Bild zusammengesetzt werden. Die Bildkompositionskomponente ist für das Auslesen der Bilddaten und für das Zusammenfügen des Ergebnisbildes verantwortlich. Wie die Netzwerkschicht muss auch diese Komponente in der Lage sein, vorhandene Hardware wie beispielsweise Sepia [MHS99] oder Lightning-2 [SEP01] anzusprechen.

- Replikation und Synchronisation: Diese Komponente ist für die Verteilung des Szenengraphen und des Grafikkontexts zuständig. Aufbauend auf der Netzwerkschicht werden alle Änderungen des Szenengraphen auf alle beteiligten Rechner in einem Cluster verteilt. Um den Grafikkontext und den Szenengraphen auf eine einheitliche Weise

zwischen den Rechnern im Cluster austauschen zu können, werden für die Speicherung des Grafikkontexts die gleichen Mechanismen eingesetzt wie bei der Speicherung von Werten im Szenengraph.

- Konfiguration: Die Konfigurationskomponente ist dafür zuständig, den Grafikkontext mit allen Informationen über den Cluster zu versorgen. Über diese Komponente wird auch der Algorithmus zur parallelen Bildberechnung ausgewählt und die Projektionssysteme konfiguriert.

Der hier vorgestellte Entwurf ermöglicht es, die Aufgabe der Bildberechnung sehr flexibel innerhalb eines Clusters zu verteilen. Da auf allen Instanzen des Cluster-Systems das gleiche Szenengraphensystem mit identischen Funktionseinheiten eingesetzt wird, kann ein sehr hoher Wiederverwendungsgrad erreicht werden. Für den Betrieb einer Applikation im Cluster sind keine applikationsspezialisierten Server erforderlich. Bei der Implementierung eines parallelen Algorithmus muss der Grafikkontext erweitert werden. Da auf jedem Rechner ein vollständiges Szenengraphensystem zur Verfügung steht, beschränkt sich die Implementierung eines Cluster-Algorithmus auf die Kommunikation zwischen den Rechnern.

3.2 Grafikkontext

Der Grafikkontext liefert bei der Bildberechnung alle Informationen, die nicht im Szenengraphen gespeichert sind. Üblicherweise werden in einem Szenengraphen keine Informationen gespeichert, die sich direkt auf die eingesetzte Software- und Hardwareumgebung zur Bildberechnung beziehen. Leider hat sich für die Bereitstellung dieser Informationen noch kein Standardverfahren etabliert. Beispielsweise werden die Bildschirmauflösung meist durch das Betriebssystem, die Position des Darstellungsfensters durch die Applikation und das Bildberechnungsverfahren durch das Szenengraphensystem bestimmt. Da bei der Bildberechnung in einem Cluster viele weitere Parameter zur Beschreibung der Bildberechnung erforderlich sind, wird für deren Verwaltung der Grafikkontext als Verwaltungsinstanz eingeführt.

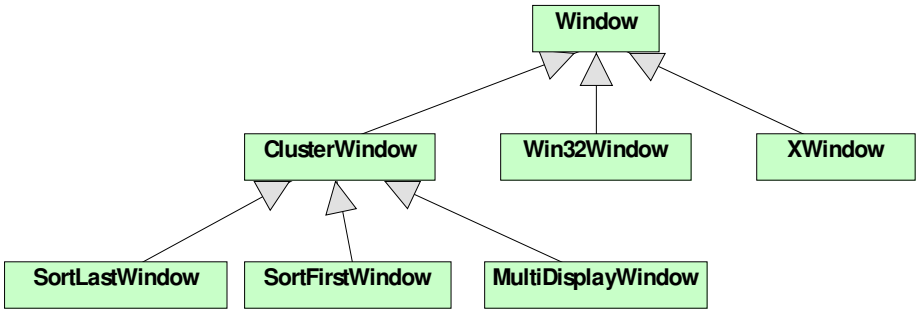

Abbildung 3-5: Datenstrukturen des grafischen Kontexts

Der Grafikkontext verwaltet Datenstrukturen zu Steuerung der Bildberechnung. Er beschreibt, wie der Szenengraph in ein 2D-Bild umgewandelt wird und wo das Bild angezeigt werden soll. In

Anlehnung an eine 2D-Benutzerschnittstelle wird die Datenstruktur als *Window* bezeichnet. Für die Darstellung unter dem Betriebssystem Windows wird die *WIN32Window*-Datenstruktur und für die Darstellung unter UNIX die *XWindow*-Datenstruktur verwendet. Für die Darstellung in einem Cluster kommen die Strukturen *SortFirstClusterWindow* und *SortLastClusterWindow* hinzu. Für neue Hardware oder neue Bildberechnungsalgorithmen können weitere Datenstrukturen hinzugefügt werden.

In einer Cluster-Umgebung wird der Grafikkontext auf alle Rechner im Cluster verteilt. Wenn jedoch auf allen Rechnern der gleiche Szenengraph von jeweils einer Instanz des gleichen Szenengraphensystems bearbeitet wird, müssten alle Rechner das gleiche Bild berechnen. Da dies für die parallele Bildberechnung sinnlos wäre, muss ein Mechanismus eingeführt werden, mit dem es dem replizierten Cluster-Grafikkontext möglich ist zu unterscheiden, auf welcher Instanz er gerade ausgeführt wird. Hierfür wird eine Server-ID eingeführt. Beim Starten der Szenengraphensysteme wird jedem System ein symbolischer Name zugeordnet. Über diesen symbolischen Namen baut das Szenengraphensystem des Clients eine Verbindung zu allen Server-Systemen auf. Hierbei wird jedem Server eine eindeutige ID zugewiesen. Der Cluster-Grafikkontext, der auf jeden Rechner im Cluster repliziert wird, kann nun mit Hilfe der Server-ID auf jedem Server einen anderen Bildbereich darstellen oder einen anderen Teil der Szene berechnen.

Der folgende Abschnitt beschreibt exemplarisch einige parallele Bildberechnungsaufgaben und wie diese mit dem hier vorgestellten Entwurf umgesetzt werden können.

3.3 Anwendungsszenarien

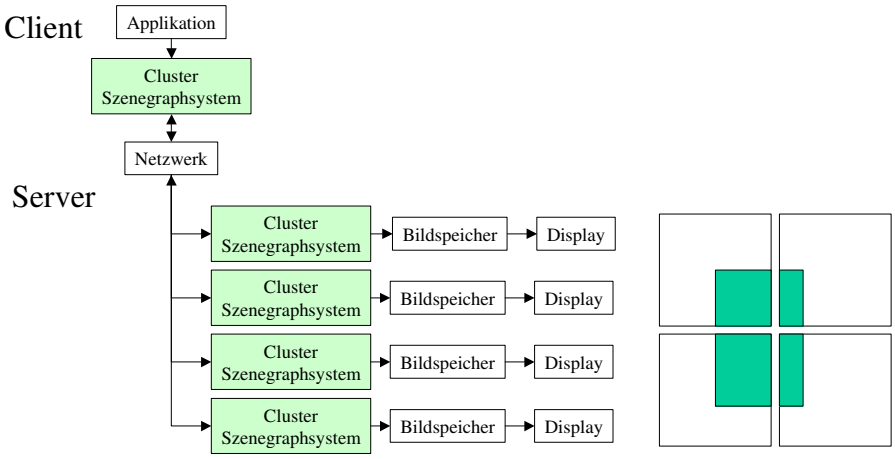

Abbildung 3-6: Konfiguration für ein Tiled-Display mit vier Projektoren

Um den Einsatz eines verteilten Szenengraphensystems in einem Cluster für unterschiedliche Aufgaben aufzuzeigen, wurden einige Anwendungsszenarien herausgesucht und entsprechende Konfigurationen abgebildet. Es werden drei unterschiedliche Szenarien untersucht. Das erste Szenario kann auf alle Projektionssysteme angewandt werden, bei denen jeweils ein Rechner einen

Teil der Projektion steuert. Dies kann beispielsweise, wie in Abbildung 3-6 gezeigt, eine gekachelte Projektion aus 2x2-Projektoren sein. Die Konfiguration besteht aus fünf Rechnern. Der Client, auf dem die Applikation ausgeführt wird, projiziert mit Hilfe von vier weiteren Rechnern ein zusammenhängendes Bild auf eine Leinwand. Der Grafikausgang des Clients wird in dieser Konfiguration nicht benötigt. Auch die Komponente zur Bildkomposition wird bei dieser Art der Projektion nicht benötigt. Mit dieser Konfiguration lassen sich beliebige Projektionssysteme mit mehreren Projektoren betreiben. Dies reicht von einfachen passiven Stereo-Projektionen mit zwei Projektoren über eine Cave mit bis zu zwölf Projektoren bis hin zu hochauflösenden Displays mit dutzenden von Projektoren. Die parallele Bildberechnung beschränkt sich darauf, dass jeder Rechner im Netzwerk nur den Ausschnitt berechnet, der auf der angeschlossenen Projektion sichtbar ist. Eine Lastverteilung ist hierbei nicht zwingend notwendig.

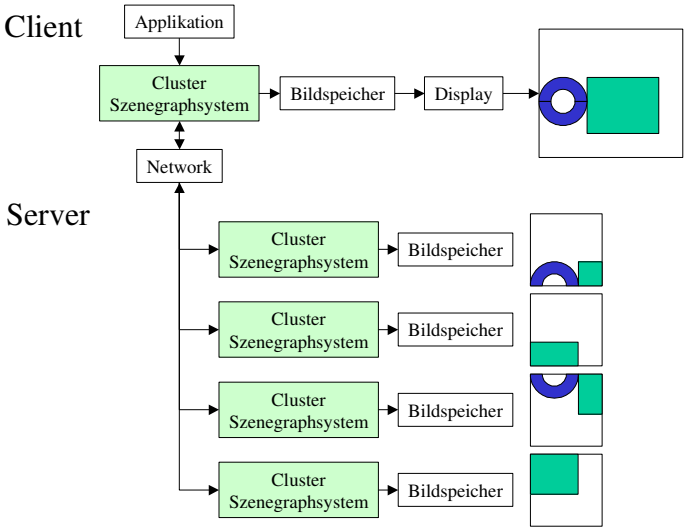

Abbildung 3-7: Konfiguration für eine Sort-First-parallele Bildberechnung

Die beiden folgenden Szenarien zeigen die Konfiguration eines Clusters, bei der alle Rechner gemeinsam ein einziges Bild berechnen. Abbildung 3-7 zeigt die Konfiguration für eine parallele Bildberechnung auf der Basis von Sort-First. Der Bildbereich wird hierbei in nicht überlappende Bereiche aufgeteilt. Jeder Bereich wird von einem anderen Rechner berechnet. Die fertigen Teilbilder werden nach der Bildberechnung aus dem Bildspeicher ausgelesen und an den Client gesendet. Dieser fügt alle Teilbilder zu einem vollständigen Ergebnisbild zusammen. Im Gegensatz zum vorhergehenden Szenario wird hier auch die Komponente zur Bildkomposition benötigt. An den Servern muss kein Projektionssystem angeschlossen werden. Die Bilddaten werden aus dem Bildspeicher ausgelesen und mit Hilfe der Bildkompositionskomponente weiterverarbeitet. Die Teilbilder können entweder über ein Netzwerk versendet oder mit der Hilfe spezieller Hardware wie beispielsweise Sepia [MHS99] oder Lightning-2 [SEP01] versendet werden. Bei Lightning-2 ist auch das Auslesen des Bildspeichers nicht erforderlich, da die Bilddaten über den digitalen Videoausgang der Grafikkarte ausgelesen werden. Die Aufgabe der Bildkompositions-komponente

beschränkt sich hierbei auf das Konfigurieren der Kompositionshardware. Das Ergebnis der Bildkomposition wird über das Display des Client-Rechners angezeigt.

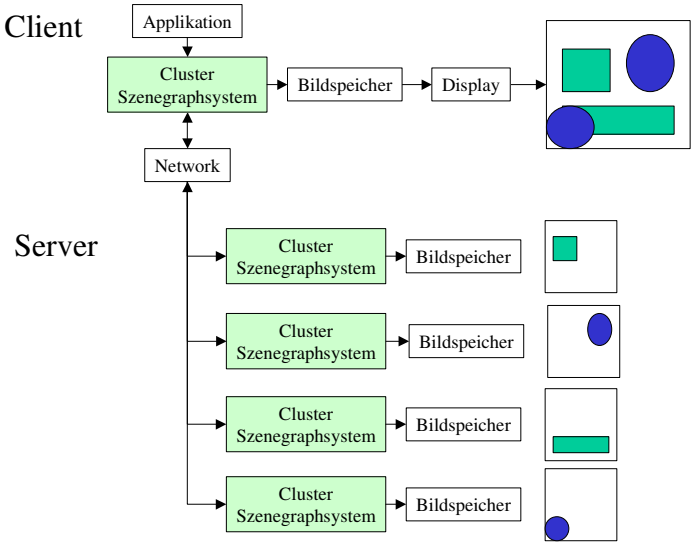

Abbildung 3-8: Konfiguration für eine Sort-Last-parallele Bildberechnung

Abbildung 3-8 zeigt ebenfalls eine Konfiguration für die parallele Bildberechnung. Jeder Rechner im Cluster übernimmt die Aufgabe, einen Teil der Szene zu berechnen. Bei dieser auf Sort-Last aufbauenden Art der parallelen Bildberechnung berechnet jeder Server ein Bild in voller Auflösung, das jedoch nur einen Teil der Szene enthält. Die Bilder aller Server müssen mit Hilfe des Tiefenwertes jedes Pixels zu einem einzigen Ergebnisbild zusammengefügt werden. Hierbei fallen sehr große Datenmengen an. Die Bildkomposition für diesen Fall muss sehr effizient mit der zur Verfügung stehenden Netzwerkbandbreite umgehen. In einem späteren Kapitel wird ausführlich auf die Bildkomposition für Sort-Last-basierte Verfahren eingegangen. Abbildung 3-8 zeigt eine Konfiguration, bei der lediglich die Server an der Bildberechnung beteiligt sind. Es ist jedoch auch möglich, dass der Client ebenfalls einen Teil der Szene berechnet. In vielen Fällen ist es jedoch effizienter, wenn das Szenengraphensystem auf dem Client lediglich die Lastverteilung übernimmt. Während die Server ein Bild berechnen, kann der Client bereits die Lastverteilung für das nächste Bild vornehmen.

Die hier beschriebenen Szenarien sollen zeigen, welche Funktionen der Client und welche Funktionen die Server übernehmen können. Die Funktionen können auch beliebig gemischt werden. Beispielsweise ist es auch möglich, eine Stereoprojektion mit zwei Projektoren so zu betreiben, bei der das Bild jedes Projektors wiederum von mehreren Rechnern parallel berechnet wird.

3.4 Konfiguration

Die im letzten Abschnitt aufgeführten Szenarien zeigen, wie viele unterschiedliche Arten der parallelen Bildberechnung in einem Cluster möglich sind. Die Aufgabe, ein Cluster-System für

eine bestimmte Aufgabe zu konfigurieren, kann sehr komplex sein. Um einem Anwender diese Aufgabe zu erleichtern, wird eine Komponente zur Erstellung und Verwaltung von Konfigurationen vorgesehen. Diese Komponente ist nicht direkt Teil des Szenengraphensystems. Vielmehr wird es in dieser Arbeit als separates System zur Konfiguration des Szenengraphensystems betrachtet. Diese Aufteilung hat folgende Vorteile:

- Die Trennung der Konfiguration von der Applikation ermöglicht es, eine Applikation vollständig von der verwendeten Cluster-Hardware und den verwendeten Cluster-Algorithmen zu trennen.

- Konfigurationen sind unabhängig von der Applikation. Beispielsweise lässt sich eine einmal erstellte Konfiguration für eine Cave in allen Applikationen wiederverwenden.

- Wenn neue Cluster-Algorithmen in das Szenengraphensystem aufgenommen werden, muss lediglich die Konfigurationskomponente angepasst werden.

Die Konfigurationskomponente muss eine Vielzahl von Parametern verwalten und dem Anwender in einer verständlichen und einfach zu handhabenden Form zugänglich machen. Hierzu wird eine grafische Benutzerschnittstelle vorgesehen. Die folgende Aufstellung zeigt, welche Parameter bei der Bildberechnung im Cluster verwaltet werden müssen.

- Identifikation der Rechner im Cluster.
- Auswahl der Netzwerkprotokolle.
- Auswahl des Algorithmus zur parallelen Bildberechnung.
- Projektionsmatrix aller angeschlossenen Projektionen. Beispielsweise muss in einer Cave der gleiche Szenengraph aus verschiedenen Blickrichtungen dargestellt werden.
- Der sichtbare Bildausschnitt, falls für ein Projektionssystem nicht der volle Bildbereich genutzt wird.
- Die Überlappung bei gekachelten Projektionen.
- Überblendungstextur für Edge Blending per Software.
- Parameter, um den Farbraum der angeschlossenen Projektionssysteme anzugleichen.
- Parameter für die geometrische Entzerrung der Projektion.
- Auswahl des Bildkompositionsverfahrens und der Bildkompositions-Hardware.
- Auswahl einer Methode, um Server im Cluster zu starten.

Auf die Bedeutung der einzelnen Parameter wird in einem späteren Kapitel genauer eingegangen. Die hier aufgeführte Liste ist nicht vollständig und kann mit der Integration neuer Hardware oder neuer Algorithmen für die Bildberechnung im Cluster eine Vielzahl weiterer Parameter beinhalten. Die Auflistung soll die Bedeutung einer benutzerfreundlichen Komponente zur Konfiguration eines Clusters verdeutlichen. Ohne diese Komponente wäre eine vollständige Integration der Unterstützung von Clustern in einen Szenengraphen zwar möglich, jedoch für einen Anwender kaum noch zu handhaben.

Eine Trennung der Konfiguration von der Applikation bedeutet nicht zwangsläufig, dass ein Anwender zwei unterschiedliche Bedienoberflächen starten muss. Die im Rahmen dieser Arbeit entwickelte Konfigurationskomponente kann als eigenständiges Programm oder als integrierter Konfigurationsdialog in einem VR-System angesprochen werden.

3.5 Zusammenfassung

Das hier vorgestellte Konzept sieht eine vollständige Integration der Bildberechnung im Cluster in ein Szenengraphensystem vor. Als Basis für diese Entwicklung dient das Szenengraphensystem OpenSG [Rot02,RRV03,RVR04]. Es wird eine strikte Trennung der Applikation von der eingesetzten Hardware und den parallelen Bildberechnungsalgorithmen vorgesehen. Dies vereinfacht die Applikationsentwicklung und ermöglicht die nachträgliche Integration neuer Hardware und neuer Verfahren ohne eine Änderung an bestehenden Applikationen. Im Gegensatz zu einem Szenengraphen für eine lokale Bildberechnung werden einige neue Komponenten eingeführt. Diese haben klar definierte Aufgaben. Die klare Aufgabentrennung ermöglicht es, den größten Teil eines Szenengraphen für eine lokale Bildberechnung unverändert für die Bildberechnung im Cluster zu verwenden. Wird der Szenengraph um neue Funktionen wie beispielsweise die Darstellung von Volumendaten oder die Ausführung von Fragment- oder Pixel-Shadern erweitert, muss keine der Komponenten geändert werden, die für das Clustering zuständig sind. Der gewählte Ansatz ermöglicht prinzipiell alle Formen der parallelen Bildberechnung in einem Cluster.

4 Kommunikation in einem Cluster

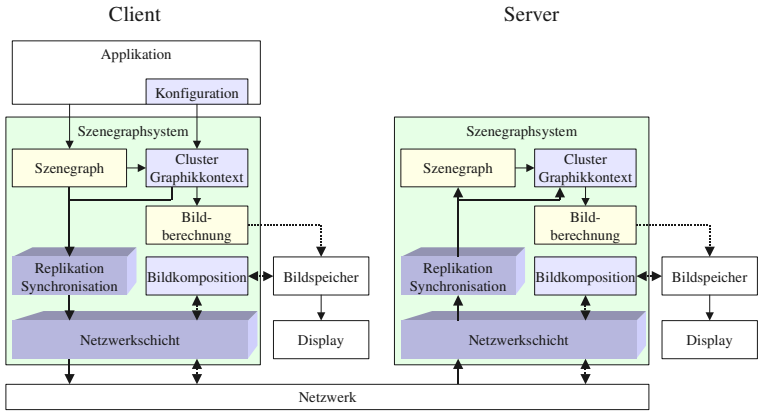

Abbildung 4-1: Replikation und Netzwerkprotokolle

Dieses Kapitel beschreibt Mechanismen zur Synchronisation eines Szenengraphen in einem Cluster-System und den Entwurf eines erweiterbaren Kommunikations-Frameworks für alle Kommunikationsaufgaben, die bei der parallelen Bildberechnung anfallen.

4.1 Replikation und Synchronisation des Szenengraphen

Ziel dieser Arbeit ist die Entwicklung eines Szenengraphensystems für die Bildberechnung in einem Cluster. Hierbei wird die Funktionalität des bestehenden Szenengraphensystems OpenSG entsprechend erweitert. Es wird versucht, bestehende Mechanismen unverändert zu nutzen. Falls dies nicht möglich ist, sollen Erweiterungen möglichst nahtlos in bestehende Konzepte integriert werden. Bei der Synchronisation des Szenengraphen müssen alle Veränderungen der Datenbasis in der Client-Applikation an alle Rechner im Cluster übertragen werden. Auf der Basis einer konsistenten Beschreibung der Szene erfolgt anschließend die parallele Berechnung der Darstellung. Ein ähnliches Problem besteht auch bei der Synchronisation mehrerer Threads auf einem einzelnen Rechner. In OpenSG ist für die Synchronisation des Szenengraphen zwischen mehreren Threads bereits ein Mechanismus vorhanden. Dieser Mechanismus wird für die Verwendung in einem Cluster um Netzwerkfunktionalitäten erweitert [VRB02, RVR04].

In OpenSG werden alle Inhalte des Szenengraphen in so genannten Feldcontainern gespeichert. Jeder Knoten des Szenengraphen wird mit Hilfe eines Feldcontainers abgebildet. Die Parameter der Knoten werden in Feldern gespeichert. Felder sind die kleinste Einheit, die bei der Synchronisation betrachtet werden. Jede Änderung des Szenengraphen wird in einer Änderungsliste vermerkt. Änderungen sind hierbei das Erzeugen neuer Knoten, das Verändern von Feldinhalten und das Löschen von Knoten. Zusätzlich werden in OpenSG Referenzzähler eingesetzt, um Knoten, die nicht mehr benötigt werden, zu erkennen und automatisch zu entfernen. Änderungen dieser Referenzzähler werden ebenfalls in der Änderungsliste gespeichert.

OpenSG kann jedem parallel laufenden Thread eine eigene Sicht auf den Szenengraphen bereitstellen. Zu einem bestimmten Zeitpunkt lassen sich jeweils zwei Sichten miteinander synchronisieren. Durch diesen Mechanismus kann beispielsweise ein Simulationssystem den Szenengraphen lesen und verändern, ohne dass eine parallel durchgeführte Visualisierung inkonsistente Inhalte verarbeitet. Eine Sicht auf den Szenengraphen wird in OpenSG als *Aspect* bezeichnet. Um das Konzept auch in einer Cluster-Umgebung einsetzen zu können, wird es um einen Remote-Aspect erweitert. Ein bereits bestehender lokaler Aspect wird für die Synchronisation zwischen Threads eines Rechners verwendet, während ein Remote-Aspect die Synchronisation von Threads auf unterschiedlichen Rechnern unterstützt.

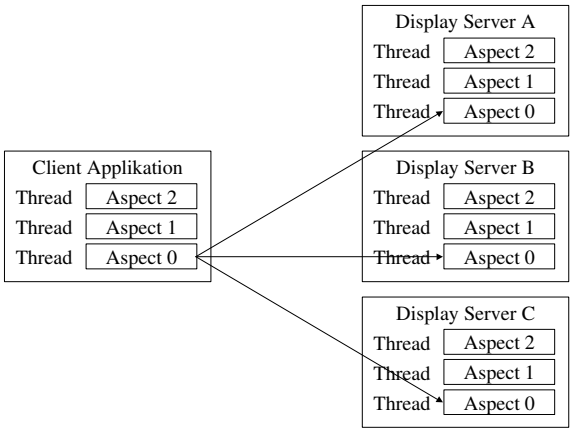

Abbildung 4-2: Synchronisation im Cluster

Bei der Synchronisation über ein Netzwerk werden die Änderungsliste und alle veränderten Feldinhalte serialisiert und als Datenstrom an alle Rechner in einem Cluster übertragen. Für jeden neuen Feldcontainer wird vermerkt, auf welchem Rechner er erzeugt wurde. Bei der Übertragung von Feldinhalten zwischen zwei Rechnern muss im Datenstrom eindeutig der Knoten des Szenengraphen kodiert werden, der die Feldänderungen empfangen soll. Für die Synchronisation zwischen zwei Threads auf einem Rechner wird zur Identifikation die Feldcontainer-ID verwendet. Diese ID wird durch eine fortlaufende Nummerierung aller Feldcontainer erzeugt. Sie kann jedoch nicht zur Identifizierung von Feldcontainern auf unterschiedlichen Rechnern verwendet werden, da die Reihenfolge der Knotengenerierung von Rechner zu Rechner abweichen kann. Um dieses Problem zu lösen, wird im Synchronisationsdatenstrom die ID verwendet, die auf dem sendenden Rechner gültig ist. Enthält der Datenstrom einen neuen Feldcontainer, dann speichert der empfangende Rechner die entfernte und die lokale Container-ID. Bei allen Modifikationen wird mit dieser Information die entfernte in eine lokale ID umgesetzt. Durch die Übertragung der Container-ID des Senders können die Daten unverändert an viele Empfänger gesendet werden. Jeder Empfänger ist für die Übersetzung der empfangenen ID in eine lokal gültige Identifikation verantwortlich.

Mit Hilfe der in OpenSG bereits vorhandenen Mechanismen kann die Synchronisation in einem Cluster so integriert werden, dass für einen Anwender kein Unterschied zwischen einer lokalen und

einer entfernten Synchronisation besteht. Ein Programm, das auf OpenSG aufbaut und für die Bildberechnung auf einem Rechner mit mehreren Grafikpipelines entwickelt wurde, kann sehr einfach für die Bildberechnung in einem Cluster angepasst werden.

4.2 Replikation des Grafikkontexts

Der grafische Kontext beschreibt die Art und Weise, wie die Umsetzung des Szenengraphen in ein zweidimensionales Abbild geschieht. Zu dieser Beschreibung gehören in einem Szenengraphensystem, das auf einem einzelnen Rechner ausgeführt wird, die Bildschirmauflösung, Farbtiefe, Position der Darstellung auf dem Bildschirm und Parameter für das Glätten der Kanten. Soll ein Szenengraphensystem in einem Cluster eingesetzt werden, muss die Beschreibung des grafischen Kontexts um zusätzliche Parameter erweitert werden. Dies sind z.B. die Namen der Rechner im Cluster, der verwendete Parallelisierungsalgorithmus oder Parameter für die Bildkomposition. Bei der parallelen Bildberechnung arbeiten alle Rechner gemeinsam an der Visualisierung des Szenengraphen. Die Koordination der einzelnen Rechner erfolgt mit Hilfe des grafischen Kontexts. Damit dieser auf allen Rechnern zur Verfügung steht, muss er, wie auch der Szenengraph, im Cluster verteilt werden. Änderungen der Parameter müssen zwischen allen Rechnern synchronisiert werden.

Für die Synchronisation des grafischen Kontexts ist es nicht notwendig, einen eigenen Mechanismus einzuführen. Die oben beschriebenen Synchronisation des Szenengraphen basiert auf der Verwaltung von Feldern und Feldcontainern. Damit dieser Mechanismus auch zur Synchronisation des grafischen Kontexts verwendet werden kann, müssen dessen Parameter in Feldern und Feldcontainern gespeichert werden, d.h., Grafikkontext und Szenengraph basieren auf den gleichen Datenstrukturen. Dies hat den zusätzlichen Vorteil, dass auch für das Laden und Modifizieren von Szenengraph und Grafikkontext die gleichen Mechanismen eingesetzt werden können. Hierdurch wird die Anwendungsentwicklung wesentlich vereinfacht.

Bisher wurde lediglich beschrieben, welcher Datenstrom für die Synchronisation des Szenengraphen versendet werden muss. Die folgenden Abschnitte befassen sich mit der Frage, wie dieser und andere Datenströme effizient zwischen den Rechnern eines Clusters verteilt werden können.

4.3 Entwurf einer erweiterbaren Kommunikationsschicht

PC-Cluster werden meist als lose gekoppelte Systeme aufgebaut, die keinen gemeinsamen Hauptspeicher besitzen. Alle Informationen müssen zwischen den Knoten des Clusters über ein Netzwerk übertragen werden. Wie in [RaH01] beschrieben, kann auch in einem PC-Cluster ein gemeinsamer Hauptspeicher simuliert werden. Hierbei werden mit Hilfe der MMU (Memory Management Unit) entfernte Speicherbereiche in den lokalen Adressraum eines Rechners eingeblendet. Die Speicherbereiche werden bei Bedarf von Rechner zu Rechner über das Netzwerk gesendet. Mit diesem Ansatz lassen sich Applikationen, die für Mehrprozessor-Systeme mit gemeinsamem Speicher entwickelt wurden, mit geringem Entwicklungsaufwand auf Cluster-

Systeme übertragen. Die Anbindung des Hauptspeichers über ein Netzwerk ist im Vergleich zu lokalen Speicherzugriffen jedoch sehr langsam. Dies betrifft sowohl die Übertragungsbandbreite als auch die Verzögerungszeiten beim Speicherzugriff. Der Einsatz von Shared-Memory in einem Cluster führt jedoch häufig zu einer schlechten Skalierung der Performanz. Der Kommunikationsaufwand kann hierbei so groß werden, dass einzelne Anwendungen in einem Cluster langsamer ausgeführt werden als auf einem einzelnen Rechner.

Die Datenrate zwischen den Knoten eines Clusters ist im Vergleich zur Verarbeitungsgeschwindigkeit moderner Prozessoren sehr gering. Die Kommunikation zwischen den einzelnen Rechner beansprucht meist einen großen Teil der Verarbeitungszeit. Aus diesem Grund muss bei der Entwicklung neuer Algorithmen, die in einem Cluster eingesetzt werden sollen, großer Wert auf die effiziente Kommunikation gelegt werden.

Eine exakte Kontrolle des Nachrichtenaustauschs lässt sich auf der Basis von Shared-Memory nur schwer erreichen. Aus diesem Grund wird in Cluster-Systemen üblicherweise ein Datenaustausch auf der Basis von Nachrichten realisiert. Hierbei werden häufig standardisierte Verfahren eingesetzt. Sehr weit verbreitet ist beispielsweise MPI (Message Passing Interface). MPI stellt umfangreiche Methoden zur Nachrichtenübermittlung bereit. Die zugrundeliegende Hardware ist durch das Applikations-Interface von der Applikation getrennt. Es existieren Implementierungen für Ethernet, Myrinet [BCF95] und andere proprietäre Kommunikationssysteme. Eine weitere Bibliothek zur Parallelisierung in einem Cluster ist PVM [GBD94]. Auch hier erfolgt die Kommunikation über einen Nachrichtenaustausch. Allerdings hat die Bedeutung von PVM in den letzen Jahren stark abgenommen.

Bei der Entwicklung von OpenSG wurde bewusst auf den Einsatz von Middleware wie z.B. MPI oder PVM verzichtet. Durch die Entwicklung einer eigenen Kommunikationsschicht besteht die Möglichkeit, sehr effiziente und an die Bedürfnisse eines verteilten Szenengraphen angepasste Mechanismen zu implementieren. Da OpenSG auch als Plattform für weitere Forschungsarbeiten dienen soll, müssen flexible und bei Bedarf auch austauschbare Mechanismen bereitgestellt werden. Beim Einsatz von vorhandenen Bibliotheken besteht diese Möglichkeit nur sehr eingeschränkt.

Bei der Entwicklung eines Kommunikations-Frameworks für OpenSG stehen folgende Anforderungen im Mittelpunkt:

- Neue Hardware für die Kommunikation in einem Cluster soll mit möglichst geringem Entwicklungsaufwand in OpenSG integrierbar sein.
- Änderungen des Szenengraphen sollen möglichst schnell auf viele Rechner eines Clusters verteilt werden.
- Die Kommunikation soll auch zwischen verschiedenen Prozessor-Architekturen und unterschiedlichen Betriebssystemen möglich sein.
- Die Applikations-Schnittstelle soll unabhängig von der verwendeten Hardware sein.

- Neben der Kommunikationsgeschwindigkeit sollen die Ressourcen CPU und Speicherbandbreite geschont werden.

Betrachtet man die aktuelle Entwicklung schneller Cluster-Netzwerke, wird deutlich, dass abgesehen von Ethernet noch kein weit verbreiteter Standard existiert. Um auf neue Entwicklungen schnell reagieren zu können, muss es möglich sein, unterschiedliche Hardware in das Kommunikations-Framework zu integrieren. Dies wird durch die Einführung einer abstrakten Kommunikationsschicht erreicht. Eine Applikation kann damit zur Laufzeit die schnellste verfügbare Hardware auswählen. Änderungen am Quellcode der Applikation sind hierbei nicht erforderlich.

Der folgende Abschnitt beschreibt den Entwurf und die Implementierung eines erweiterbaren Frameworks für die Kommunikation zwischen Rechnern in einem Cluster. Das Framework soll die oben beschriebenen Anforderungen erfüllen und primär für die Synchronisation des Szenengraphen eingesetzt werden. Es soll jedoch auch möglich sein, einzelne Funktionen für andere Zwecke einzusetzen. Beispielsweise kann die Kodierung und Dekodierung der Daten auch zum Schreiben des Szenengraphen oder zur Synchronisation in einem VR-System für mehrere Benutzer eingesetzt werden.

4.3.1 Das Verbindungs-Framework

In OpenSG steht der Szenengraph im Mittelpunkt aller Aktivitäten. Aus diesem Grund beruht die in dieser Arbeit vorgestellte Kommunikation zwischen den Rechnern eines Clusters im Wesentlichen auf der Synchronisation des Szenengraphen. Eine Applikation in einem Cluster besteht aus einem Client und mehreren Servern. Der Client stellt das Interface zum Anwender zur Verfügung. Alle Änderungen des Szenengraphen, die aus dieser Interaktion resultieren, werden an alle Server übertragen.

Da die Synchronisation auf der Basis des Szenengraphen erfolgt, muss kein spezieller Server für jede Anwendung entwickelt werden. Der Szenengraph beschreibt die darzustellende Szene vollständig. Aus diesem Grund kann die gleiche Implementierung des Servers alle Anwendungen bedienen. Bei der Kommunikation zwischen Client und Server ist kein applikationsspezifisches Wissen erforderlich. Für alle Applikationen kann der gleiche OpenSG-basierte Server eingesetzt werden. Dies vereinfacht sowohl die Administration als auch die Software-Entwicklung. Ein Anwendungsentwickler kann sich auf die Erfordernisse der Applikation konzentrieren. Ein applikationsspezifischer Cluster-Server muss nicht entwickelt werden.

Änderungen des Szenengraphen werden in OpenSG in so genannten Änderungslisten gespeichert. Diese Listen enthalten aus Platzgründen nicht die Werte einer Änderung, sondern nur einen Verweis auf den geänderten Wert. Wie in der Arbeit von Dirk Reiners [Rei02] ausführlich beschrieben, werden neben Feldänderungen auch Strukturveränderungen des Szenengraphen gespeichert. Da die Informationen der Änderungslisten exakt die Informationen sind, die bei der

Synchronisation des Szenengraphen in einen Cluster erforderlich sind, ist es naheliegend, diese vorhandenen Datenstruktur zu nutzen. In OpenSG wird in Multiprozessor-Systemen der Szenengraph mit Hilfe der Änderungsliste zwischen unterschiedlichen Threads synchronisiert. Erweitert man dieses Konzept für den Einsatz in einem Cluster, so erhält man lokale und entfernte Threads, zwischen denen eine Synchronisation durchgeführt werden kann. Für den Anwendungsentwickler ist das Clustering lediglich eine Erweiterung der bereits existierenden Multithreading-Funktionalität.

Um die oben beschriebenen Anforderungen an die Synchronisation des Szenengraphen zu erfüllen, muss die Änderungsliste möglichst gleichzeitig an viele Server verteilt werden. Damit die Daten zwischen unterschiedlichen Rechnerarchitekturen ausgetauscht werden können, müssen sie in ein plattformunabhängiges Format umgewandelt werden. Um eine Unabhängigkeit von der eingesetzten Hardware zu erreichen, wird im Rahmen dieser Arbeit ein erweiterbares Framework entwickelt. Im Folgenden wird das abstrakte Interface dieses Frameworks beschrieben. Im Anschluss daran werden konkrete Implementierungen auf der Basis von TCP/IP und UDP/IP vorgestellt. Implementierungen für Myrinet oder Infiniband können auf einfache Weise in das Framework integriert werden. Die vorliegende Arbeit konzentriert sich jedoch auf Ethernet als Kommunikationssystem. Insbesondere wird bei der Beschreibung konkreter Implementierungen auf die besonderen Anforderungen von Gigabit-Ethernet eingegangen.

Das im Folgenden beschriebene Framework dient neben der effizienten Synchronisation des Szenengraphen auch der Kommunikation zwischen den Knoten des Clusters. Beispielsweise werden mit Hilfe des Frameworks auch Bilddaten und andere Zwischenergebnisse der parallelen Bildberechnung übertragen. Hierbei kann es sich im Falle der Bilddaten um sehr große Datenmengen handeln. Für die Übertragung großer Datenmengen muss die zur Verfügung stehende Bandbreite mit Hilfe von Lese- und Schreibpuffern möglichst gut ausgenutzt werden. Es kann jedoch auch der Fall eintreten, dass sehr viele kleine Nachrichten ausgetauscht werden müssen. Hier steht nicht der maximale Durchsatz im Mittelpunkt, sondern eine möglichst kurze Verzögerungszeit. Das im Folgenden beschriebene Framework stellt Mechanismen zur Verfügung, die es ermöglichen, beiden Anforderungen gerecht zu werden.

4.3.2 Binäre Kodierung des Szenengraphen

Eine zentrale Klasse des neu entwickelten Kommunikations-Frameworks ist die *BinaryDataHandler*-Klasse. Sie stellt ein abstraktes Interface zur binären Kodierung des Szenengraphen zur Verfügung. Ein *BinaryDataHandler* wandelt Feldinhalte des Szenengraphen in einen binären sequentiellen Datenstrom um. Hierbei werden einzelne Daten zu größeren Blöcken zusammengefasst, um eine effiziente Nutzung der Netzwerkhardware zu ermöglichen. Für jede Netzwerkhardware muss eine konkrete Klasse von *BinaryDataHandler* abgeleitet werden. Es müssen konkrete Implementierungen für die abstrakten Methoden *read* und *write* bereitgestellt werden, d.h., es müssen lediglich Methoden für das Lesen und Schreiben der Daten implementiert werden. Die Kodierung und Dekodierung der Daten erfolgt in der Basisklasse.

Da es in einigen Netzwerken günstig sein kann, große Datenblöcke gesondert zu behandeln, können zusätzlich die Methoden *readBlock* und *writeBlock* bereitgestellt werden. Mit diesen Methoden können mehrere Lese- und Schreiboperationen zusammengefasst werden. Bei paketorientiertenNetzwerken können aufeinanderfolgende Schreiboperationen gebündelt werden, um eine optimale Paketgröße für die konkrete Hardware zu bilden. Damit dies von einer Applikation hardwareunabhängig eingesetzt werden kann, muss jede konkrete Implementierung einen oder mehrere Speicherbereiche zur Pufferung der Daten bereitstellen. Die Anzahl und Größe dieser Puffer kann beliebig an die Anforderungen der eingesetzten Hardware angepasst werden.

Der *BinaryDataHandler* stellt für jeden in OpenSG existierenden Datentyp eine Methode für das Kodieren und Dekodieren der Daten in ein plattformunabhängiges Format bereit. Diese Konvertierung erfordert Rechenzeit, die eingespart werden kann, wenn Sender und Empfänger zur internen Speicherung der Daten das gleiche Format verwenden. In einem homogenen Cluster können Daten ohne eine vorhergehende Kodierung zwischen den Rechnern ausgetauscht werden. Beim Aufbau einer neuen Netzwerkverbindung wird das Speichermodell beider Rechner verglichen. Speichern beide Seiten die Daten intern im Little-Endian- oder Big-Endian-Format, werden die Daten ohne eine vorhergehende Konvertierung gesendet. Lediglich von Big-Endian zu Little-Endian und umgekehrt ist eine Konvertierung in ein plattformunabhängiges Netzwerkformat erforderlich. Weitere Konvertierungen wie beispielsweise die Anpassung von Zeichensätzen von ASCII nach EPSDIC oder die Konvertierung von Fließkommazahlen ist leicht in das bestehende System zu integrieren. Da dies jedoch bei keiner der von OpenSG unterstützten Plattformen erforderlich ist, werden in der aktuellen Implementierung keine weiteren Konvertierungen unterstützt.

4.3.3 Optimierte Pufferung der Daten

Eine weitere wichtige Funktion des *BinaryDataHandlers* ist die effiziente Pufferung der Daten. In fast allen Kommunikationssystemen steigt die Datenrate mit der Größe der übertragenen Datenblöcke. Aus diesem Grund werden die zu sendende Daten in einem Puffer gesammelt und erst beim Erreichen einer für die konkrete Hardware optimalen Datenmenge gesendet. Diese Pufferung findet sowohl auf der Sender- als auch auf der Empfängerseite statt.

Durch die Pufferung kann die Übertragung vieler kurzer Nachrichten sehr stark beschleunigt werden. Sollen jedoch sehr große Datenmengen wie z.B. Texturdaten übertragen werden, müssen diese zuerst in einen Puffer kopiert werden. Aufgrund der Größe der Daten wird der Inhalt des Puffers unmittelbar nach dem Kopiervorgang gesendet. In diesem Fall führt der Kopiervorgang zu einer Verschlechterung des Durchsatzes im Vergleich zu einer ungepufferten Datenübertragung. Zusätzlich verlangsamt die für das Kopieren erforderliche Speicherbandbreite andere Prozesse in einem Rechnersystem.

Im hier vorgestellten Ansatz wird das Problem durch die Einführung eines Schwellwertes gelöst. Nachrichten, deren Länge den Schwellwert unterschreiten, werden in einem Puffer gesammelt. Nachrichten, deren Größe den Schwellwert überschreiten, werden ungepuffert gesendet. Der Schwellwert kann für jede Netzwerkhardware individuell bestimmt werden. Die gepufferte und ungepufferte Übertragung kann jedoch nicht ohne weiteres miteinander gemischt werden. Der Empfänger wäre nicht mehr in der Lage, einen korrekten Datenstrom zu rekonstruieren. Dieses Problem kann auf zwei Arten gelöst werden. Zum einen kann vor dem Senden einer großen Nachricht der Inhalt des Puffers übertragen werden. Dies führt jedoch dazu, dass eventuell ein sehr kurzer Datenblock an die Hardware übergeben wird. Ein besseres Ergebnis lässt sich durch eine verzögerte Übertragung großer Datenblöcke erreichen. Für einen großen Datenblock wird ein Flag von der Größe eines Bytes in den Puffer geschrieben. Erst wenn der Puffer vollständig gefüllt ist, wird dessen Inhalt gesendet. Anschließend werden alle zurückgehaltenen großen Datenblöcke übertragen. Mit diesem Vorgehen lassen sich sowohl kleine als auch umfangreiche Feldinhalte während der Synchronisation effizient übertragen.

Mit Hilfe des oben erwähnten ein Byte großen Flags lässt sich das gepufferte Schreiben großer Datenblöcke erzwingen. Dies ist dann erforderlich, wenn die zu schreibenden Feldinhalte nicht verzögert geschrieben werden können, da sie nur vorübergehend im Hauptspeicher gehalten werden. Ein Beispiel hierfür sind komprimierte Texturdaten. Um die Datenmenge bei der Synchronisation zu reduzieren, können Texturen in komprimierter Form übertragen werden. Hierzu werden die Bilddaten unmittelbar vor der Übertragung in ein komprimierendes Bildformat wie z.B. JPEG umgewandelt. Nach der Übergabe der Daten an den *BinaryDataHandler* wird das komprimierte Bild wieder gelöscht. In diesem Fall können die Daten nicht verzögert gesendet werden, da sie zum Zeitpunkt des verzögerten Sendens nicht mehr zur Verfügung stehen. In diesem speziellen Fall werden auch große Datenblöcke gepuffert versendet.

4.3.4 Ein binäres Dateiformat

Die Hauptaufgabe des *BinaryDataHandlers* besteht in der effizienten Kodierung und Pufferung der Änderungen des Szenengraphen. Er bietet jedoch auch alle Funktionen, die für das Lesen und Schreiben eines binäres Dateiformats erforderlich sind. Um die Funktionsweise und Effizienz der entwickelten Verfahren zu testen, wird ein neues Dateiformat in OpenSG integriert. Die Effizienz bei der Kodierung und Dekodierung der Daten lässt sich anschaulich am Laden einer großen Szene zeigen. Zuerst wird der Szenengraph aus einer FHS-Datei (Dateiformat der Firma VR-Com) erzeugt. Anschließend wird der Szenengraph mit Hilfe des *BinaryDataHandlers* in ein binäres Format umgewandelt. Vergleicht man nun das Laden beider Dateiformate, so zeigt sich, dass das binäre Format bis zu 100 mal schneller geladen werden kann. Dies lässt darauf schließen, dass der *BinaryDataHandler* auch in einem Netzwerk effizient eingesetzt werden kann. Der Test hat auch gezeigt, dass alle Informationen des Szenengraphen zuverlässig kodiert und dekodiert werden können. Mit einigen Erweiterungen wird der *BinaryDateHandler* auch für das Lesen und Schreiben des binären Dateiformats OSB in OpenSG verwendet. Hierbei werden dem Datenstrom neben reinen Feldwerten auch Metainformationen hinzugefügt. Mit Hilfe dieser Metainformationen

können binäre Dateien von verschiedene OpenSG-Versionen gelesen werden. Dies ist auch dann noch möglich, wenn die Struktur des Szenengraphen zwischen den Versionen geändert wurde.

4.3.5 Abstrakte Netzwerkverbindungen

Die Klasse *BinaryDataHandler* stellt ein einheitliches Interface zum Lesen und Schreiben in unterschiedlichen Netzwerkumgebungen bereit. Die Art der Verbindung sowie der Verbindungsaufbau werden noch nicht berücksichtigt. Aus diesem Grund wird aufbauend auf dem *BinaryDataHandler* ein erweiterbares Framework entwickelt. Ziel ist es, ein hardwareunabhängiges Interface bereitzustellen. Die Applikation soll keinen Zugriff auf konkrete Netzwerk-Implementierungen haben. Hierdurch wird erreicht, dass in jeder Applikation die Netzwerkschicht ausgetauscht werden kann. Die Auswahl eines bestimmten Netzwerks erfolgt mit Hilfe einer Factory. Der Factory wird ein String als Identifikation für ein bestimmtes Netzwerk übergeben. Ist für dieses Netzwerk eine konkrete Implementierung vorhanden, dann wird ein Objekt erzeugt, das von der Klasse *Connection* abgeleitet ist. Die Applikation hat lediglich Zugriff auf das allgemeine Interface der *Connection*-Klasse.

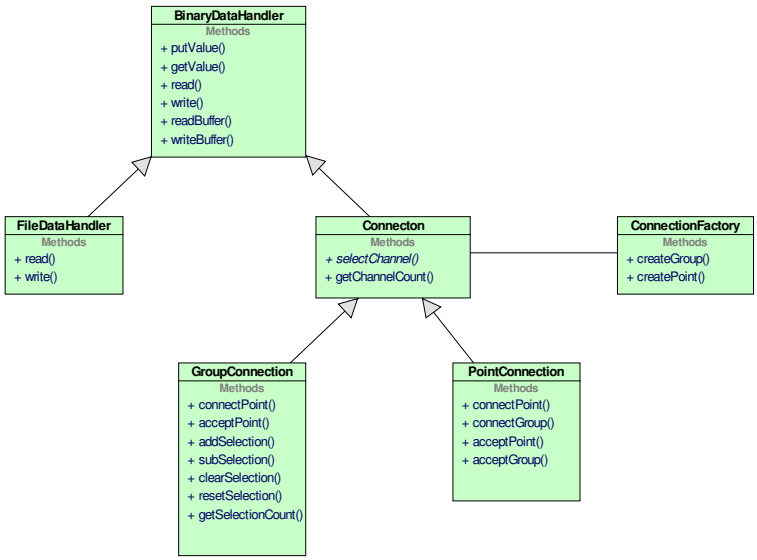

Abbildung 4-3: Klassendiagramm der Netzwerkverbindungen

Bestimmte Netzwerke erwarten neben Adressinformationen zusätzliche Parameter. Beispielsweise die Auswahl eines bestimmten Gerätes, falls mehrere Geräte das gleiche Netzwerk unterstützen, oder zusätzliche Parameter, die Einfluss auf die Datenübertragung haben. Da das Interface unabhängig von der zugrundeliegenden Hardware sein soll, müssen diese Parameter über das allgemeine Interface übergeben werden. Im Rahmen dieser Arbeit werden drei Kategorien von Parametern unterschieden.

- Interface-Parameter: Diese beschreiben Eigenschaften des Interfaces und ermöglichen eine Auswahl, falls dies erforderlich ist.

- Zielparameter: Diese Parameter beeinflussen die Datenübertragung zum Empfänger. Ein Beispiel hierfür ist die Multicast-Adresse bei einer Multicast-Netzwerkverbindung.

- Sonstige Parameter: Je nach Hardware können beliebig viele Parameter zur Verfügung stehen. Neben den zwei oben beschriebenen Kategorien hat die Applikation jedoch keinen Einfluss auf weitere Parameter. Alle Parameter, die nicht über das Interface übergeben werden können, müssen von der konkreten Implementierung automatisch optimiert werden. Beispiele für diese Parameter sind Puffergrößen oder Quality-of-Service-Angaben.

Für die Synchronisation des Szenengraphen müssen die gleichen Daten an viele Empfänger verteilt werden. Es wird daher ein Verbindungstyp benötigt, mit dessen Hilfe ein Datenpaket automatisch an mehrere Empfänger verteilt werden kann. Je nach Netzwerk muss diese Funktion entweder durch mehrere Einzelverbindungen oder mit Hilfe einer Mehrpunktverbindung realisiert werden. Beispiele für Mehrpunktverbindungen sind Broadcast und Multicast in einem Ethernet-Netzwerk.

Bei der Synchronisation des Szenengraphen sind zwei Typen von Verbindungen zu unterscheiden. Der Sender schickt die Änderungen des Szenengraphen mit Hilfe einer Mehrpunktverbindung. Das Interface der hierbei erforderlichen Funktionen wird in der Klasse *GroupConnection* definiert. Alle Empfänger lesen die ankommenden Daten aus einer einzigen Quelle. Das hierbei erforderliche Interface wird in der Klasse *PointConnection* definiert.

Theoretisch bestünde die Möglichkeit, die Klasse *PointConnection* als Spezialfall der Klasse *GroupConnection* mit einem einzigen Empfänger zu betrachten. Es hat sich jedoch gezeigt, dass eine Aufteilung mehr Spielraum für Optimierungen bietet. Zusätzlich wird die Anbindung neuer Netzwerkhardware vereinfacht. In einem späteren Abschnitt wird genauer auf die Implementierung eines Multicast-Protokolls eingegangen. Bei dieser Multicast-Anbindung kann aufgrund der Aufteilung in *Point-* und *GroupConnection* sehr einfach festgelegt werden, wann Pakete per Multicast verschickt werden müssen und wann es effizienter ist, eine Punkt-zu-Punkt-Verbindung zu wählen.

Für die Synchronisation des Szenengraphen ist lediglich eine Verbindung von einer *Group-* zu einer *PointConnection* erforderlich. Für die parallele Bildberechnung in einem Cluster sind jedoch auch schnelle Punkt-zu-Punkt-Verbindungen erforderlich. Aus diesem Grund werden die Verbindungen Point ó Group und Point ó Point unterstützt.

4.3.6 Gruppenverbindungen

Ein Objekt der Klasse *GroupConnection* liest aus mehreren Quellen und schickt Daten an mehrere Empfänger. Beim Senden an mehrere Empfänger müssen die gleichen Daten zuverlässig an alle Ziele gesendet werden. Bei der Anbindung eines Netzwerks muss die Zuverlässigkeit der Datenübertragung mit Hilfe geeigneter Protokolle gewährleistet werden, falls die korrekte Datenübertragung von der Netzwerkhardware nicht garantiert wird. Da das Senden an mehrere

Empfänger für die Applikation transparent erfolgt, ist hierfür keine Erweiterung des Interfaces der Klasse *BinaryDataHandler* erforderlich. Anders verhält es sich mit dem Lesen aus mehreren Quellen. Hierfür sind spezielle Mechanismen erforderlich, um die Vermischung von Datenströmen zu verhindern. Es muss gewährleistet sein, dass erst eine komplette Nachricht aus Quelle A gelesen wurde, bevor mit dem Lesen einer Nachricht aus Quelle B begonnen wird. Hierfür ist eine von der Applikation gesteuerte Quellenselektion erforderlich. Nur der Empfänger kann den Datenstrom analysieren und feststellen, wann eine Nachricht zu Ende gelesen wurde und wann mit dem Lesen aus einer anderen Quelle begonnen werden darf. Für diese Art der Selektion muss das Interface der Klasse *BinaryDataHandler* erweitert werden.

Eine konkrete Implementierung der Selektion kann dahingehend optimiert werden, dass immer aus der Quelle gelesen wird, die das größte Datenvolumen im Eingabepuffer bereitstellt. Durch dieses Vorgehen kann das Lesen aus einer großen Anzahl von Einzelverbindungen beschleunigt werden. Da eine solche Auswahl mit steigender Anzahl an Verbindungen zeitaufwendig sein kann, wird ein Mechanismus eingeführt, mit dessen Hilfe einzelne Verbindungen aus der Selektion ausgeschlossen werden können. Die Einschränkung der Selektion kann für die Optimierung bestimmter Aufgaben herangezogen werden. Soll beispielsweise von jeder Quelle genau ein Datenpaket gelesen werden, dann kann der Lesevorgang dadurch beschleunigt werden, dass nach jedem erfolgeichen Lesen eines Datenpaketes aus einer Verbindung diese Verbindung aus den folgenden Selektionen ausgeschlossen wird. Mit abnehmender Zahl der selektierten Quellen wird bei einigen Netzwerktypen die Ausführung der Selektionsoperation schneller. Als Alternative für diese Technik könnte man für diesen speziellen Fall auch einfach jede Quelle direkt selektieren. Dies hat jedoch den Nachteil, dass nicht mehr aus der Quelle gelesen werden kann, die zuerst Daten liefert. Die Methode der variablen Selektionen ist daher der direkten Auswahl von Quellen überlegen.

4.3.7 Einzelverbindung

Objekte der Klasse *PointConnection* stellen eine Verbindung zu genau einer Datenquelle und zu genau einer Datensenke bereit. Eine *PointConnection* kann entweder mit einer *GroupConnection* oder einer *PointConnection* verbunden werden. Da Daten im Gegensatz zu *GroupConnections* lediglich aus einer Quelle gelesen werden, ist kein Interface für die Quellenselektion erforderlich. Um *Group-* und *PointConnection* jedoch auf einheitliche Weise ansprechen zu können, werden die Methoden *selectChannel* und *getSelectionCount* auch für das Interface der Klasse *PointConnection* bereitgestellt.

4.3.8 Verbindungsnetzwerke

Mit Hilfe der oben beschriebenen Einzel- und Gruppenverbindungen lassen sich Kommunikationswege zwischen den Knoten eines Clusters sehr genau an die Bedürfnisse der eingesetzten Algorithmen anpassen. Beispielsweise ist es möglich, jeden Knoten mit jedem anderen Knoten mit Hilfe zweier *PointConnections* zu verbinden. Dieses Verbindungsnetzwerk

erlaubt den einfachen Nachrichtenaustausch zwischen beliebigen Rechnern im Cluster. Theoretisch kann man mit diesem Verbindungsnetzwerk jede Kommunikationsaufgabe bewältigen. Es gibt jedoch Aufgaben, bei denen es sinnvoll ist, jeden Rechner mit Hilfe einer *GroupConnection* mit allen anderen Rechnern zu verbinden. In einer solchen Konfiguration kann jeder Knoten des Clusters schnell Nachrichten an alle Rechner verteilen oder mit einer einzigen Leseoperation auf das Eintreffen einer Nachricht von einem beliebigen anderen Rechner warten. Je nach Aufgabe können auch beide Verbindungsnetzwerke kombiniert oder sogar parallel aufgebaut werden.

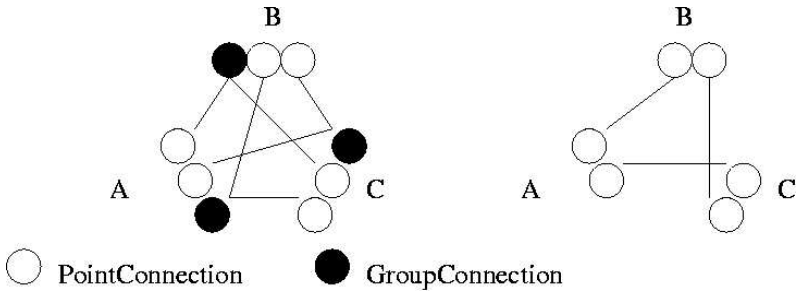

Abbildung 4-4: Verbindungsnetzwerke

Das Kommunikations-Framework ist als integraler Bestandteil in OpenSG vorgesehen. Das Framework kann hierbei nicht nur für die Umsetzung paralleler Bildberechnungsalgorithmen im Cluster verwendet werden, sondern steht auch Applikationsentwicklern für die Realisierung applikationsspezifischer Kommunikationsaufgaben zur Verfügung. Beispielsweise lässt sich die Kommunikation in Multiuser-VR-Anwendungen sehr effizient mit Hilfe des Kommunikations-Frameworks realisieren.

Die folgenden Abschnitte beschreiben die Realisierung konkreter Einzel- und Gruppenverbindungen. Es werden unterschiedliche Verbindungstypen für lokale Ethernet-Netzwerke vorgestellt. Ethernet wurde als Basis ausgewählt, da es auch in Cluster-Umgebungen die am häufigsten eingesetzte Netzwerktechnik ist.

4.4 Entwurf eines verlässlichen Multicast-Protokolls

Netzwerke auf der Basis von Ethernet bieten die Möglichkeit, eine Nachricht gleichzeitig an viele Empfänger zu versenden. Dies ist eine sehr effiziente Möglichkeit, einen Szenengraph sehr schnell auf viele Rechner zu verteilen. Ein Problem hierbei besteht darin, dass Multicast in Ethernet-Netzwerken auf der Basis einer unzuverlässigen paketorientierten Datenübertragung realisiert werden muss. Einzelne Pakete können verloren gehen, mehrfach oder in geänderter Reihenfolge beim Empfänger ankommen. Bei der Datenübertragung lassen sich mehrere Qualitätsstufen der Übertragung unterscheiden.

- Unreliable: Ein Paket kommt einmal, mehrmals oder gar nicht beim Empfänger an. Die Transportschicht leitet empfangene Pakete direkt weiter an den Empfänger.

- Unordered: Jedes Paket kommt genau einmal beim Empfänger an. Die Reihenfolge kann jedoch von der Sendereihenfolge abweichen. Ein ankommendes Paket wird sofort an den Empfänger weitergereicht, sofern dies noch nicht geschehen ist. Fehlende Pakete werden vom Sender angefordert.

- Totally ordered: Jedes Paket kommt genau einmal und in der gleichen Reihenfolge, in der sie der Sender verschickt hat, beim Empfänger an. Pakete werden erst dann an den Empfänger weitergeleitet, wenn alle vorangegangenen Pakete ausgeliefert wurden.

Für die zuverlässige Synchronisation des Szenengraphen in einem Cluster ist die Anforderung „Totally ordered“ unerlässlich. Es muss sichergestellt werden, dass jeder Empfänger exakt die Daten erhält, die vom Sender verschickt wurden. Andernfalls wäre eine Synchronisation des Szenengraphen im Cluster nicht möglich. Da bei Ethernet-Multicast einzelne Pakete verloren gehen können, müssen sich Sender und Empfänger mit Hilfe eines Protokolls darüber informieren, welche Pakete gesendet und erfolgreich empfangen wurden. Hierbei lassen sich zwei verschiedene Ansätze unterscheiden. Entweder teilt der Empfänger dem Sender nur das Ausbleiben von Paketen mit, oder der Empfänger quittiert korrekt erhaltene Pakete. Man spricht hierbei von NAK(negative-acknowledge)-basierten Protokollen oder von ACK(acknowledge)-basierten Protokollen.

4.4.1 Untersuchung bestehender Multicast-Protokolle

Leider hat sich bis heute noch kein einheitliches Protokoll für eine verlässliche Datenübertragung per Multicast etabliert. Ein Grund hierfür besteht darin, dass je nach Anwendung sehr unterschiedliche Anforderungen an ein Protokoll gestellt werden. In den folgenden Abschnitten werden unterschiedliche Ansätze bei der Realisierung eines verlässlichen Multicast-Protokolls untersucht. Aufbauend auf den hierbei gewonnenen Erkenntnissen wird anschließend ein neues Protokoll für die Kommunikation in einem Cluster entworfen. Neben den hier vorgestellten Protokollen wurden im Laufe der Jahre viele weitere Protokolle entwickelt, die jedoch keine größere Verbreitung erfahren haben [Che96,FJL97,JGJ00]. Ein Vergleich unterschiedlicher Ansätze findet sich auch in [LDY01], wobei dort nicht auf die Besonderheiten eines Clusters eingegangen wird..

4.4.1.1 ACK-basierte Protokolle

Bei einem ACK-basierten Multicast-Protokoll [FJL97,PSL97] werden Pakete vom Sender per Multicast gesendet. Alle Empfänger bestätigen den Erhalt eines Paketes mit einem kurzen Datenpaket (ACK). Der Sender kann ein Paket aus dem Sendepuffer löschen, wenn er eine positive Quittung von allen Empfängern erhalten hat. Gehen Pakete verloren, kann dies der Sender sehr einfach feststellen, da hierbei eine oder mehrere erwartete Quittungen ausbleiben. Fehlende Pakete können auch vom Empfänger erkannt werden, wenn die Pakete fortlaufend nummeriert sind und in den ankommenden Paketen Lücken in der Nummerierung auftreten.

Das Hauptproblem bei ACK-basierten Protokollen besteht in der großen Anzahl von Quittungen, die vom Sender verarbeitet werden müssen. Die Anzahl der positiven Quittungen steigt linear mit der Anzahl der Empfänger. ACKs sind im Vergleich zu Datenpaketen zwar sehr klein, doch kann es beim Senden an viele Empfänger vorkommen, dass der Sender mit ACKs überflutet wird. Ein großer Teil der zur Verfügung stehenden Netzwerk- und CPU-Ressourcen muss für die Bearbeitung der Quittungen aufgewendet werden.

ACK-basierte Protokolle haben den Vorteil, dass sehr schnell feststeht, welche Pakete von allen Empfängern korrekt gelesen wurden und somit aus dem Sendepuffer gelöscht werden können. Als Folge hiervon können die Sendepuffer sehr klein gehalten werden.

4.4.1.2 NAK-basierte Protokolle

NAK-basierte Protokolle [WMK94,KoZ96,MRW99] umgehen das Problem der ACK-Stürme durch den Verzicht auf positive Quittungen. Stattdessen werden von den Empfängern nur negative Quittungen (NAK) beim Ausbleiben von Paketen gesendet. In lokalen Netzwerken ist die Anzahl der Pakete, die nicht beim Empfänger ankommen, meist sehr gering. Daher ist die Anzahl der positiven Rückmeldungen wesentlich größer als die der negativen Meldungen. In NAK-basierten Protokollen werden dem Sender nur fehlende Pakete gemeldet. Der Sender muss also nur wenige Rückmeldungen bearbeiten.

Bei dieser Vorgehensweise ist der Sender nicht in der Lage festzustellen, wann ein Paket von allen Empfängern korrekt empfangen wurde. Aus diesem Grund müsste der Sendepuffer so groß sein, dass er alle jemals gesendeten Pakete aufnehmen kann. Da dies in realen Anwendungen nicht möglich ist, wird in NAK-basierten Protokollen meist ein Polling-Mechanismus verwendet. Wird beim Sender der Speicherplatz knapp, fordert er eine Empfangsbestätigung von allen Empfängern. Je nach Größe des Sendepuffers können diese Empfangsbestätigungen den Sender wie bei ACK-basierten Protokollen überfluten.

NAK-basierte Protokolle sind dann problematisch, wenn kein kontinuierlicher Datenstrom gesendet wird und das Ausbleiben eines Paketes schnell erkannt werden muss. Geht beispielsweise das letzte Datenpaket eines Datenblocks verloren, können weder der Sender noch der Empfänger dies feststellen. Erst wenn der Empfänger das erste Paket des darauffolgenden Datenblocks empfängt, kann er das Ausbleiben des vorhergehenden Paketes feststellen. Um dieses Problem zu beheben, werden in vielen Protokollen nach dem Senden des letzten Paketes in kleinen Zeitintervallen kurze Signalpakete gesendet.

In den letzten Jahren wurde eine sehr große Anzahl von Multicast-Protokollen für Ethernet-Multicast entwickelt. Leider hat sich bis heute jedoch kein Standard etabliert. Dies liegt hauptsächlich an den sehr unterschiedlichen Anforderungen an ein verlässliches Multicast-Protokoll. Beispielsweise wurde viel im Bereich des Video- und Audio-Broadcasting per Multicast geforscht. Bei der Datenübertragung ist eine absolut fehlerfreie Datenübertragung nicht zwingend erforderlich. Das Hauptproblem bei der Videoübertragung an viele Empfänger besteht darin,

Multicast über die Grenzen des lokalen Netzwerks hinaus zu benutzen. Die von Ethernet bereitgestellte Multicast-Funktionalität kann in WANs (Wide Area Network) meist nicht eingesetzt werden. Hier sind spezielle Protokolle und Netzwerk-Services wie z.B. MBONE [CaD92] erforderlich. Ein weiterer Schwerpunkt bei der Entwicklung von Multicast-Protokollen besteht im Bereich der Multiuser-VR-Umgebungen [MZP94, RaH96, Fun96, MaZ97, LJF97]. Hierbei ist es wichtig, dass gesendete Nachrichten bei allen Empfängern in der gleichen Reihenfolge eintreffen. Das betrifft nicht nur die Pakete eines einzelnen Senders, sondern alle Nachrichten von allen Teilnehmern in einer Multiuser-Umgebung müssen eine definierte Ordnung aufweisen. Senden beispielsweise die Teilnehmer A und B jeweils eine Nachricht, muss eindeutig geregelt sein, welche Nachricht als erstes verarbeitet wird. In Mutiuser-Umgebungen werden in der Regel nur sehr kleine Datenpakete verschickt. Eine Optimierung auf einen hohen Datendurchsatz ist meist nicht erforderlich.

Auch die Synchronisation des Szenengraphen in einer Cluster-Applikation stellt ganz spezielle Anforderungen an ein Multicast-Protokoll. Die Daten müssen sicher beim Empfänger ankommen, es müssen sehr große Datenmengen übertragen werden, und das Protokoll darf die Verzögerungszeit des Netzwerks nicht wesentlich erhöhen. Neben diesen Anforderungen bestehen auch einige Randbedingungen, die es ermöglichen, ein sehr effizientes Protokoll zu entwerfen. Cluster sind i.d.R. über ein lokales Netzwerk angebunden. Meist bestehen Cluster aus einer Ansammlung identischer Rechner mit einem einheitlichen Netzwerk-Interface. Bei der Entwicklung eines Protokolls kann davon ausgegangen werden, dass jeder Rechner die gleiche Datenmenge verarbeiten kann. Mit diesem Wissen lässt sich die Flusskontrolle in einem Protokoll wesentlich vereinfachen.

Viele Probleme bei der Datenübertragung per Multicast tauchen in den meisten Protokollen auf, unabhängig davon, für welchen Anwendungszweck sie optimiert sind. Aus diesem Grund werden im Folgenden einige Multicast-Protokolle genauer untersucht.

4.4.1.3 RMP

Das von Brian Whetten entwickelte RMP-Protokoll [WMK94] sendet Pakete per Ethernet-Multicast an alle Mitglieder einer Gruppe. Jeweils ein Gruppenmitglied übernimmt bei RMP eine Sonderrolle. Diese Sonderrolle wird über den Besitz eines Tokens geregelt. Der Besitzer des Tokens sendet für jedes empfangene Paket eine ACK-Quittung per Multicast an alle anderen Mitglieder der Gruppe und an den Sender. Die Mitglieder einer Gruppe erhalten durch dieses Vorgehen für jedes gesendete Datenpaket auch eine Quittung. Mit Hilfe dieser redundanten Information können fehlende Pakete meist schnell erkannt werden.

Obwohl der Sender ACK-Meldungen erhält, kann er noch nicht feststellen, ob alle Mitglieder der Gruppe das Paket erhalten haben. Um dies zu erreichen, wird der Token innerhalb einer Gruppe weitergereicht. Hierbei übernimmt ein Gruppenmitglied nur dann den Token, wenn es alle Pakete empfangen hat, die auch vom bisherige Besitzer des Tokens empfangen wurden. Wenn das Token einmal alle Gruppenmitglieder passiert hat und wieder beim Empfänger angekommen ist, steht

eindeutig fest, welche Pakete erfolgreich ausgeliefert wurden. Diese Pakete können aus dem Zwischenspeicher gelöscht werden. Diese ringförmige Kommunikation wird auch in dem etwas jüngeren QRMP-Protokoll eingesetzt [Che96].

Der in RMP gewählte Ansatz ermöglicht eine schnelle Erkennung von fehlenden Paketen. Das Herumreichen des Tokens benötigt in großen Gruppen jedoch sehr viel Zeit. Der Sender muss daher einen sehr großen Zwischenspeicher für die erneute Übertragung von fehlenden Paketen bereitstellen. Da RMP in einer Zeit entwickelt wurde, als 10- und 100-MBit-Netzwerke aktuell waren, stellte dies zur damaligen Zeit kein sehr großes Problem dar. Werden jedoch Netzwerke mit einer Bandbreite von mehreren Gigabit eingesetzt, sind bei dem in RMP gewählten Ansatz Puffergrößen von mehreren Megabyte erforderlich. Gigabit-Ethernet bietet eine sehr hohe Datenübertragungsrate, wobei die Verzögerungszeit jedoch vergleichbar mit der von 100-MBit-Ethernet ist. D.h., das Herumreichen des Tokens wird beim Einsatz von Gigabit-Ethernet nicht beschleunigt. Dadurch verzehnfacht sich die erforderliche Puffergröße.

RMP unterstützt verschiedene Qualitätsstufen bei der Datenübertragung. Brian Whetten gibt die Auslastung einer 100–MBit-Verbindung durch sein RMP-Protokoll mit 86% an.

4.4.1.4 RAMP

RAMP [KoZ96] ist ein NAK-basiertes Protokoll. Pakete werden gruppenweise gesendet (Burst Mode). Zwischen dem Versenden zweier Pakete vergehen maximal 0,5 Sekunden. Alle Pakete erhalten eine fortlaufende Nummer. Ein Empfänger kann sehr leicht feststellen, ob Daten verloren gegangen sind. Entweder wird das Fehlen eines Paketes an einer lückenhaften Nummerierung erkannt, oder es wird über einen Zeitraum von mehr als einer halben Sekunde kein Paket empfangen. In diesem Fall wird ein NAK-Paket an den Sender geschickt. Wenn der Sender keine Pakete mehr zu verschicken hat, muss er ein oder mehrere Idle-Pakete senden. Dies ist erforderlich, da der Empfänger sonst nicht unterscheiden kann, ob das Ausbleiben von Nachrichten eine Fehlersituation ist oder ob der Sender keine Daten mehr verschickt.

In einem RAMP-Protokoll besitzen Pakete ein Mode-Flag (Burst oder Idle) und ein ACK-Flag. Sendet der Sender ein Paket mit gesetztem ACK-Flag, erwartet er von allen Empfängern eine Bestätigung. Das ACK-Flag wird im ersten Paket einer Paketgruppe gesetzt. Alle Empfänger antworten mit der aktuell empfangenen Paketnummer. Hierdurch kann der Sender feststellen, ob während einer Sendepause die Verbindung zu einem oder mehreren Empfängern abgebrochen wurde.

Der Datendurchsatz in einem 10-MBit-Ethernet beträgt 6,5 MBit/s. In [KoZ96] wurde RAMP mit lediglich sechs Empfängern getestet. Für diese Anzahl skaliert das Protokoll fast linear.

4.4.1.5 RMTP

RMTP [PSL97] wurde speziell für Multicast im Internet entwickelt. Hierbei muss ein besonderes Augenmerk auf die Flusskontrolle, die Vermeidung von Überlastungen und die

Verzögerungszeiten gelegt werden. Das RMTP-Protokoll teilt Empfänger in Gruppen ein. Diese Gruppen werden so gewählt, dass topologisch naheliegende Empfänger zu einer Gruppe zusammengefasst werden. RMP ist ein ACK-basiertes Protokoll. Üblicherweise werden bei ACK-basierten Protokollen sehr viele Bestätigungspakete von allen Empfängern an den Sender übertragen. Um die Anzahl dieser Pakete zu minimieren, werden im RMTP-Protokoll alle Empfänger über eine Baumstruktur mit dem Sender verbunden. Jeweils nur ein Mitglied einer Gruppe von Empfängern sendet ein ACK-Paket an die nächst höhere Verästelungsstufe. Jede Gruppe besitzt ein Mitglied mit einer Sonderrolle. Dieses Mitglied schickt ACK-Meldungen an den Sender und empfängt NAK- und ACK-Meldungen der anderen Gruppenmitglieder. Der Sender kann seinen Sendepuffer löschen, wenn er von allen Gruppen eine positive Quittung erhalten hat. Dies muss jedoch nicht bedeuten, dass alle Empfänger alle Daten erhalten haben. Innerhalb einer Gruppe können Fehlersituationen häufig selbstständig behoben werden. Dieses Vorgehen erhöht die Effizienz des ACK-basierten Ansatzes wesentlich. RMTP kann auch in großen Netzwerken eingesetzt werden.

4.4.2 Eignung bestehender Protokolle für Cluster-Systeme

Ein Großteil der bestehenden Protokolle wurde für eine effiziente Übertragung in WANs und heterogenen Umgebungen entwickelt. Die Vernetzung eines Clusters besteht i.d.R. aus einheitlichen Komponenten. Viele Mechanismen, die für eine Datenübertragung in Wide-Area-Netzen erforderlich sind, spielen bei der Kommunikation in einem Cluster keine Rolle. Es ist zwar möglich, auf der Basis bestehender Protokolle eine Datenverteilung in einem Cluster durchzuführen, eine spezialisierte Lösung hat jedoch den Vorteil, dass höhere Datenraten und kürzere Verzögerungszeiten erreichbar sind. Um die speziellen Anforderung in einem Cluster aufzuzeigen, befasst sich der folgende Abschnitt mit den speziellen Anforderungen eines Multicast-Protokolls für den Einsatz in einem Cluster. Es wird untersucht, welche Struktur Netzwerke in einem Cluster besitzen und welche Randbedingungen zu berücksichtigen sind. Aufbauend auf den Erkenntnissen, die aus der Betrachtung bestehender Protokolle gewonnen wurden, wird untersucht, welche Mechanismen für den Entwurf eines effizientes Protokolls geeignet sind.

4.4.3 Randbedingungen für ein verlässliches Multicast-Protokoll

Cluster-Systeme, deren Rechner auf der Basis von Ethernet vernetzt sind, werden üblicherweise über einen Switch miteinander verbunden. Jeder Knoten des Clusters ist direkt über den Switch mit allen anderen Knoten verbunden. Übersteigt die Anzahl der Rechner die Anzahl an Ports eines Switches, können mehrere Switches eingesetzt werden. Diese werden über eine schnelle Verbindung miteinander vernetzt. Ein solches Netzwerk erfordert kein komplexes Routing. Die Suche nach dem Weg von einem Sender zu einem Empfänger wird von der Switching-Hardware übernommen.

Ein idealer Swich kann zwischen beliebigen Paaren von Ports Pakete in der vollen Bandbreite des Netzwerks parallel übertragen. Besitzt ein Switch beispielsweise 16 Ports und hat jeder Port eine

Bandbreite von 1 GBit, dann können acht mal 1 GBit parallel übertragen werden. Je nach Fabrikat teilen sich Ports eines Switches jedoch meist ein gemeinsames internes Verbindungsnetzwerk. Bei einem Cisco Catalyst 4006 beträgt die maximale interne Bandbreite 24 GBit. D.h., werden mehr als 48 Ports verwendet, kann der Switch nicht mehr alle Ports mit der vollen Bandbreite bedienen. In einer solchen Situation muss der Switch Pakete verwerfen, die schneller eintreffen als sie weiterversendet werden können. Für den Test der im Rahmen dieser Arbeit entwickelten Protokolle standen zwei Switches der Firma Cisco zur Verfügung. Ein Catalyst 4003 mit 48 Ports und ein Catalyst 4006 mit 96 Ports.

Geht man davon aus, dass ein Cluster üblicherweise aus Rechnern der gleicher Leistungsklasse zusammengesetzt ist und alle Rechner über einen Switch miteinander verbunden sind, dann lassen sich einige Aspekte beim Entwurf eines verlässlichen Multicast-Protokolls wesentlich vereinfachen. Ethernet-Switches können Multicast-Pakete an alle Ports verteilen, die sich für eine Multicast-Gruppe angemeldet haben. Eine spezielle Infrastruktur wie MBONE ist nicht erforderlich. Alle Rechner können mit der gleichen Geschwindigkeit Daten lesen und schreiben. Dies vereinfacht die Umsetzung einer Flusskontrolle. In Multicast-Protokollen, die auch über große Entfernungen eingesetzt werden sollen, können sehr große Zeitunterschiede zwischen dem Eintreffen eines Paketes bei unterschiedlichen Empfängern bestehen [CMM96]. In einem Cluster mit einem lokalen Netzwerk werden alle Multicast-Pakete nahezu zeitgleich empfangen.

Cluster-Algorithmen erfordern meist eine sehr hohe Bandbreite insbesondere dann, wenn Bilddaten übertragen werden sollen. Meist werden hierfür Gigabit-Netzwerke eingesetzt. 10-Gigabit-Netwerke stehen bereits zu Verfügung, werden jedoch aufgrund der hohen Kosten nur selten eingesetzt. Soll ein Multicast-Protokoll 10 Gigabit an Datentransfer bewältigen, ist ein NAK-basierter Ansatz sehr problematisch. Da der Sender nur einen begrenzten Sendepuffer bereithalten kann, müssen häufig Empfangsbestätigungen eingeholt werden. Dies hebt den Vorteil des NAK-basierten Ansatzes jedoch wieder auf. Es müssen nicht nur die fehlenden Pakete mit einem NAK quittiert werden, sondern immer dann, wenn der Sendepuffer zur Neige geht, müssen auch ACKs von den Empfängern gesendet werden, damit der Sender wieder Pufferplatz freigeben kann. Aus diesem Grund wird für das hier vorgestellten Mutlicast-Protokoll ein ACK-basierter Ansatz gewählt.

Einige Protokolle gehen davon aus, dass die Rate der Pakete, die nicht beim Empfänger ankommen, sehr gering ist. Dies trifft auch für die meisten 100MBit-Netzwerke zu. Leider erreicht die Rate der nicht zugestellten Pakete in einem Gigabit-Netzwerk einen so hohen Wert, dass hohe Datenraten nur dann erreicht werden können, wenn fehlende Pakete schnell erkannt und erneut gesendet werden. In [BCD04] werden Gigabit-Netzwerke unter verschiedenen Randbedingungen untersucht. Die dort gewonnenen Erkenntnisse können auch für die Infrastruktur, die für diese Arbeit zur Verfügung stand, nachvollzogen werden. Demnach sind Fehlerrate und Datentransferrate bei Multicast stark von der eingesetzten Hardware abhängig.

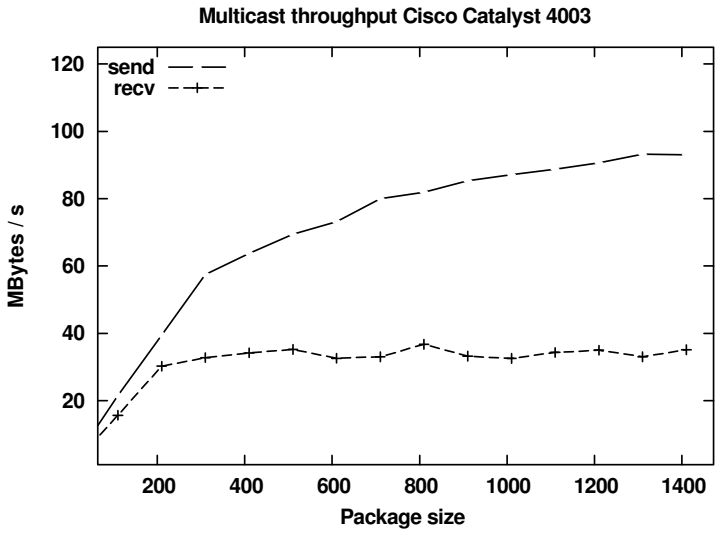

Abbildung 4-5: Maximaler Multicast-Durchsatz für Casio Catalyst 4003

Abbildung 4-5 zeigt die Datenübertragungsraten für Multicast auf dem Casio-Catalyst-4003-Switch. Bei diesem Versuch werden vom Sender Pakete mit unterschiedlichen Längen so schnell wie möglich versendet. Es ist deutlich zu sehen, dass die Senderate mit steigender Paketgroße zunimmt. Dies liegt hauptsächlich daran, dass das Betriebssystem jedes Paket einzeln bearbeiten muss und diese Verarbeitungszeit weitgehend unabhängig von der Paketlänge ist. Erst ab einer Paketgröße von etwa 300 Bytes macht sich die Sendegeschwindigkeit der Gigabit-Netzwerkkarte bemerkbar. Der Test wurde bis zur maximalen Paketgröße bei Ethernet von 1500 Bytes durchgeführt. Größere Pakete werden vom Betriebssystem in kleinere Pakete zerlegt und vom Empfänger wieder zu einem großen Paket zusammengefügt. Geht eines dieser Teilpakete verloren, so werden vom Betriebssystem alle anderen Teile des Pakets verworfen. Es ist daher nur dann sinnvoll, große Pakete zu verschicken, wenn sichergestellt ist, dass die Wahrscheinlichkeit für eine fehlerhafte Übertragung sehr gering ist. Betrachtet man die Empfangsrate, so ist zu erkennen, dass beim Empfänger nur ein Teil der gesendeten Pakete ankommt. Beim Cisco Catalyst 4003 werden maximal 30 MBytes/s vom Empfänger korrekt empfangen. Um zu zeigen, dass es sich hierbei um eine Limitierung des Switches handelt, wurde der gleiche Test mit Unicast wiederholt. Der Arbeitsaufwand hierfür ist für das Betriebssystem identisch. Bei der Übertragung per Unicast kommen die Daten vollständig beim Empfänger an.

Pakete gehen in lokalen Netzen immer dann verloren, wenn ankommende Pakete nicht mehr in einen Empfangspuffer geschrieben werden können. Ein anderer Grund sind fehlerhafte Übertragungen. Diese sind jedoch sehr selten und treten meist nur bei defekten Netzwerkkabeln oder anderen technischen Störungen auf. Das Betriebssystem verwirft ankommende Pakete, wenn die Applikation nicht in der Lage ist, aus dem Empfangspuffer schnell genug zu lesen. Unter Linux ist es möglich, die Anzahl der verworfenen Pakete auszulesen. Mit dieser Information lässt sich untersuchen, an welcher Stelle der Übertragung ein Engpass besteht.

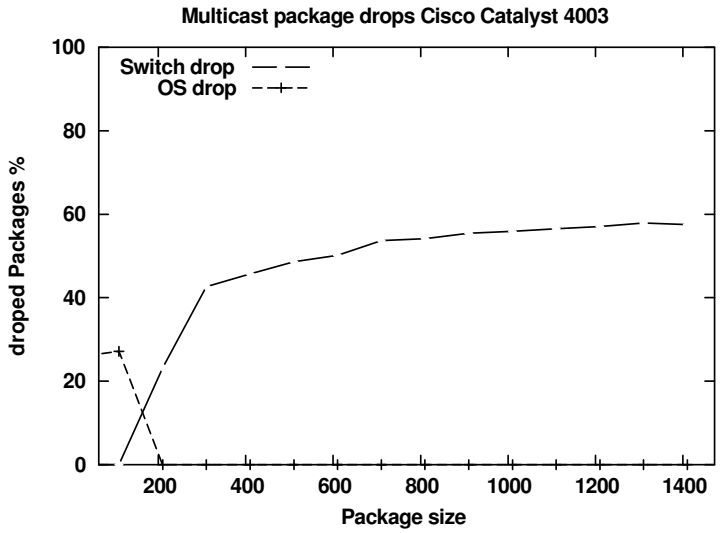

Abbildung 4-6. Fehlerhafte Übertragung bei Multicast

Abbildung 4-6 zeigt die prozentuale Fehlerrate bei der Übertragung von Multicast mit maximaler Sendebandbreite. Es zeigt sich, dass bei Paketen mit weniger als 200 Bytes hauptsächlich das Betriebssystem für eine hohe Fehlerrate verantwortlich ist. Versucht man eine Gigabit-Verbindung mit Paketen von 200 Bytes Länge auszulasten, dann müssen etwa 500.000 Pakete pro Sekunde verschickt werden. Dies übersteigt offensichtlich die Leistungsfähigkeit des eingesetzten Linux-Kernels. Für die hier aufgeführten Tests wurde der Linux-Kernel 2.4.19 eingesetzt. In [BCD04] wird ausgeführt, dass in neueren Versionen des Kernels effizientere Mechanismen implementiert wurden. Unabhängig davon werden jedoch bei Paketgrößen über 200 Bytes fast alle Pakete nicht mehr vom Betriebssystem verworfen. Wenn Datenpakete nicht mehr auf dem Zielrechner ankommen und nicht vom Betriebssystem verworfen wurden, müssen sie auf dem Weg über den Switch verloren gegangen sein.

Neben einem Catalyst-4003-Switch von Cisco stand auch ein Catalyst 4006 des gleichen Herstellers zur Verfügung. Tests mit diesem Switch zeigen, dass die oben beschriebene hohe Fehlerrate von der Switching-Hardware abhängig ist. Wie in Abbildung 4-7 zu sehen, ist der Catalyst 4006 in der Lage, Pakete in der Geschwindigkeit weiterzureichen, in der sie gesendet werden können. Die Fehlerrate ist hierbei sehr gering. Im Hinblick auf die Entwicklung eines effizienten verlässlichen Multicast-Protokolls müssen daher Beschränkungen des Switches berücksichtigt werden.

4.4.4 Fehlererkennung und Fehlerbehebung

Die wichtigste Entscheidung beim Design eines neuen Multicast-Protokolls ist die Festlegung auf einen Ansatz mit positiven (ACK) oder einen mit negativen Quittungen (NAK). Für die schnelle Synchronisation des Szenengraphen ist eine hohe Bandbreite erforderlich. Um die Ressourcen aktueller Netzwerkhardware ausschöpfen zu können, muss ein verlässliches Multicast-Protokoll Bitraten von einem Gigabit und in naher Zukunft auch 10 Gigabit bewältigen können. Diese hohe

Datenrate erfordert bei Verwendung eines NAK-basierten Ansatzes sehr große Sendepuffer. Ein weiterer Nachteil des reinen NAK-basierten Ansatzes ist, dass der Empfänger keine Kenntnis darüber besitzt, wann alle Empfänger die gesendeten Daten erhalten haben. Aus diesem Grund wird im Rahmen dieser Arbeit ein ACK-basierter Ansatz für die verlässliche Synchronisation des Szenengraphen per Multicast gewählt.

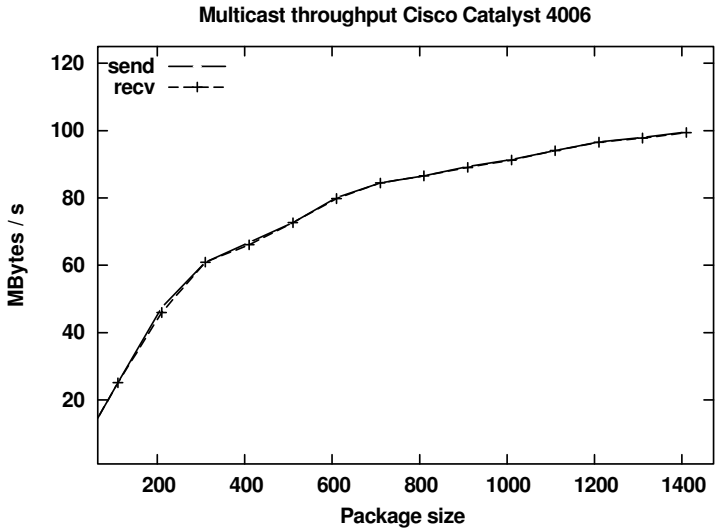

Abbildung 4-7: Multicast-Durchsatz mit Cisco Catalyst 4006

ACK-basierte Protokolle lassen sich meist nicht effizient in großen Netzwerken einsetzen. Grund hierfür ist, dass mit steigender Anzahl an Empfängern die Anzahl der ACK-Pakete rapide zunimmt. Das Lesen und Verarbeiten dieser Pakete durch den Sender bildet ab einer gewissen Anzahl an Empfängern den Flaschenhals der Datenübertragung. Um dieses Problem in den Griff zu bekommen, werden im hier vorgestellten Ansatz die Empfänger in Gruppen aufgeteilt. Jede Gruppe besitzt ein Mitglied, das erst dann ein ACK-Paket an den Empfänger sendet, wenn es von allen anderen Mitgliedern der Gruppe eine positive Quittung erhalten hat. Dieser Ansatz wird auch in RMTP [PSL97] verwendet, um Gruppierungen aufgrund der Netzwerktopologie vorzunehmen. In einem lokalen Netzwerk, in dem alle Rechner über einen Switch verbunden sind, spielt die Topologie bei der Gruppeneinteilung keine Rolle. Die Empfänger können beliebig in Gruppen mit gleicher Mitgliederzahl eingeteilt werden. In dem hier vorgestellten neuen Protokoll wird eine zweistufige Hierarchie verwendet. Die Empfänger werden in Gruppenmitglieder und Gruppenverwalter eingeteilt. Gruppenmitglieder schicken für jede korrekt empfangene Paketgruppe ein ACK-Paket an den Gruppenverwalter. Der Gruppenverwalter schickt für alle Pakete, die er empfangen hat und für die er von jedem Gruppenmitglied eine positive Quittung empfangen hat, ein ACK-Paket an den Sender. Mit dieser zweistufigen Hierarchie lässt sich die Anzahl der ACK-Pakete auf $n^{1/2}$ reduzieren. In einem Cluster aus 256 Knoten muss der Empfänger und jeder Gruppenverwalter 16 ACKs pro Paketgruppe verarbeiten. Da ACK-Pakete aus wenigen Bytes bestehen, ist diese Anzahl ohne Probleme zu bewältigen. Soll das Protokoll auch für mehrere tausend Empfänger eingesetzt werden, muss lediglich eine weitere Hierarchie eingefügt werden. Dies lässt sich mit einem geringen Aufwand realisieren.

4.4.5 Flusskontrolle

In einem lokalen Netzwerk kann die Flusskontrolle für ein Multicast-Protokoll sehr einfach gehalten werden. Jeder Empfänger ist in der Lage, die Menge an Daten zu verarbeiten, die der Sender verschickt. Alle Empfänger erhalten die ankommenden Daten fast zeitgleich. Die Flusskontrolle muss in einer solchen Umgebung lediglich sicherstellen, dass an keiner Stelle des Übertragungsweges ein Empfangspuffer überläuft. Der Sender verwaltet hierfür einen Ringpuffer mit Paketen, die noch nicht von allen Empfängern bestätigt wurden. Erst wenn alle Empfänger ein Paket bestätigt haben, wird der Platz im Ringpuffer freigegeben, und das nächste Paket kann versendet werden. Die Größe des Ringpuffers wird so gewählt, dass ein Pufferüberlauf selten auftritt. Messungen haben ergeben, dass bei einer Größe des Ringpuffers von 128 KByte eine gute Übertragungsgeschwindigkeit und sehr geringe Paketverluste durch einen Pufferüberlauf erreicht werden können. Stellt man die Größe des Ringpuffers so ein, dass garantiert kein Überlauf stattfinden kann, sinkt die Übertragungsrate. Dies liegt daran, dass ein Folgepaket erst dann verschickt wird, wenn das vorangegangene Paket positiv von allen Empfängern quittiert wurde. Während der Sender auf die Quittungen aller Empfänger wartet, bleibt die Bandbreite des Netzwerks ungenutzt. Eine optimale Auslastung des Netzwerks wird nur dann erreicht, wenn das Senden der Daten und die Übertragung der ACK-Pakete parallel erfolgt.

4.4.6 Optimierung für positive Quittungen

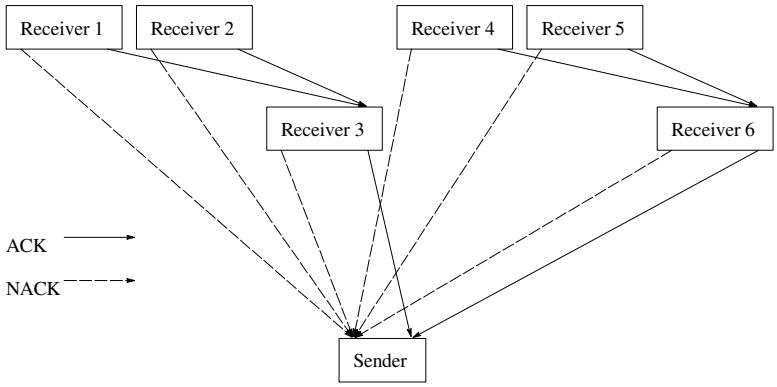

Abbildung 4-8: Bündelung positiver Quittungen

Wie oben beschrieben, wird die Anzahl der ACK-Pakete durch eine Gruppierung der Empfänger reduziert. Eine weitere Reduktion lässt sich erreichen, wenn ein Empfänger nicht jedes ankommende Paket quittieren muss. Ein Empfänger muss beispielsweise ein ankommendes Paket nicht quittieren, wenn bereits weitere Pakete im Empfangspuffer stehen. Das Lesen aus dem Empfangspuffer ist im Vergleich zur Übertragung über das Netzwerk sehr schnell. Es ist daher sinnvoll, zunächst das nächste Paket aus dem Empfangspuffer auszulesen und erst dann, wenn der Empfangspuffer geleert ist, ein ACK-Paket zu senden. Allerdings kann es hierbei passieren, dass der Sender erst relativ spät eine Bestätigung für die gesendeten Pakete erhält. Da der Sender nur dann neue Pakete senden kann, wenn er bestätigte Pakete aus dem Ringpuffer gelöscht hat, kann

sich das verzögerte Senden der ACK-Pakete negativ auf die Übertragungsrate auswirken. Um dies zu verhindern, unterdrückt der Empfänger nur so lange das Senden von Quittungen, bis er die Hälfte der Daten empfangen hat, die in den Ringpuffer des Senders passen. Damit ist gewährleistet, dass die Quittung noch rechtzeitig beim Sender ankommt, bevor der Sendepuffer vollständig gefüllt ist.

Bei einem Cluster kann in der Regel davon ausgegangen werden, dass in allen Rechnern Netzwerkhardware der gleichen Leistungsklasse vorhanden ist. Man kann daher annehmen, dass ein Empfänger ankommende Daten mit der gleichen Geschwindigkeit lesen kann, wie sie von einem beliebiger Sender mit maximaler Geschwindigkeit versendet werden können. Aus diesem Grund wird bei der Übertragung ähnlich wie bei RAMP [KoZ96] ein Burst-Mode verwendet. Der Sender versendet alle Pakete im Ringpuffer so schnell wie möglich. Das Senden wird erst dann unterbrochen, wenn der Ringpuffer mit unbestätigten Paketen gefüllt ist und auf eintreffende ACK-Pakete gewartet werden muss.

4.4.7 Asynchrone Sende- und Empfangswarteschlangen

Eine Applikation kann in vielen Fällen die zu versendenden Daten wesentlich schneller liefern als sie von der Netzwerkhardware versendet werden können. Würde man die Daten direkt bei der Übergabe von der Applikation über das Netzwerk senden, wäre die komplette Applikation an die Übertragungsgeschwindigkeit des Netzes gekoppelt. In einem langsamen Netzwerk würde eine Applikation sehr viel Zeit mit dem Warten auf freie Netzwerkbandbreite verbringen. Dies kann vermieden werden, wenn das Bereitstellen der Daten durch die Applikation und das Senden über das Netzwerk entkoppelt werden. Hierfür wird im hier vorgestellten Ansatz die Datenübertragung in einen eigenen Thread ausgelagert. Dieser läuft asynchron zur Applikation und kümmert sich um das Senden und die komplette Protokollverarbeitung mit Flusskontrolle und Fehlerkorrektur. Applikation und Sende-Thread werden über eine Warteschlange miteinander verbunden. Der Sende-Thread geht in ein blockierendes Warten über, wenn keine Pakete mehr in der Warteschlange stehen und alle gesendeten Pakete positiv quittiert wurden. Damit verbraucht der Sende-Thread keine Rechenzeit, wenn die Applikation keine Daten verschickt. Die Applikation kann so lange Pakete in die Warteschlange schreiben, bis diese eine bestimmte Länge erreicht hat. Erst dann wird die Applikation in einen Wartezustand versetzt.

Ein ähnlicher Mechanismus wird auch für den Empfänger vorgesehen. Auch beim Lesen von Daten ist es sinnvoll, die Applikation vom eigentlichen Lesen zu trennen. Dadurch können Daten empfangen werden, auch wenn die Applikation mit anderen Aufgaben beschäftigt ist. Die ankommenden Daten werden in eine Empfangswarteschlange gestellt. Die Applikation kann zu einem beliebigen Zeitpunkt die Daten auslesen. Der Lesevorgang wird nur dann angehalten, wenn die Warteschlange eine vorgegebene Größe überschreitet.

Wenn eine Applikation das Eintreten bestimmter Ereignisse an viele Empfänger mitteilen will, kommt es meist nicht auf eine hohe Bandbreite an, sondern auf ein möglichst schnelles Versenden

der Mitteilung. In einem solchen Fall kann sich die asynchrone Datenübertragung negativ auswirken. Die Mitteilung muss zuerst in die Warteschlange gestellt werden. Anschließend muss vom Betriebssystem ein Wechsel des Threads durchgeführt werden. Erst jetzt kann die Meldung über das Netzwerk versendet werden. Auf der Empfängerseite ist ebenfalls ein Wechsel des Threads erforderlich. Für eine Datenübertragung mit geringer Latenz wäre es besser, wenn die Daten ohne einen Wechsel des Threads gesendet würden. Um dies zu erreichen, wird eine bestimmte Sende- und eine bestimmte Empfangssituation gesondert behandelt. Schreibt die Applikation ein Paket in eine Sendewarteschlange, die bisher leer war, dann wird das Paket sofort an das Netzwerk übergeben. Zusätzlich wird es in die Warteschlange geschrieben. Der Sende-Thread überprüft für dieses Paket lediglich die korrekte Auslieferung an alle Empfänger. Das eigentliche Senden erfolgt ohne einen Wechsel des Threads. Ähnlich wird beim Empfänger verfahren. Versucht die Applikation aus einer leeren Lesewarteschlange zu lesen, dann wird direkt auf eintreffende Pakete reagiert. Hierbei muss sichergestellt werden, dass ein Paket nicht zweimal verarbeitet wird. Ein Paket kann direkt im Applikations-Thread empfangen und zusätzlich noch vom Lese-Thread in die Warteschlange gestellt werden. Dies lässt sich jedoch mit Hilfe der Paketnummer auf einfache Weise vermeiden. Das hier vorgestellte Übertragungsmodell ermöglichst es, die Vorteile der synchronen und die Vorteile der asynchronen Datenübertragung zu kombinieren.

Bei der Datenübertragung über ein unsicheres Medium muss sichergestellt werden, dass alle Empfänger alle Daten erhalten, und es muss auch sichergestellt werden, dass ein Empfänger möglichst schnell erfährt, dass er ein bestimmtes Datenpaket nicht empfangen hat. Der korrekte Empfang aller Pakete lässt sich anhand der ankommenden ACK-Pakete beim Sender sicherstellen. Erhält der Sender innerhalb einer bestimmten Zeitspanne keine Bestätigung, dann sendet er eine ACK-Anforderung an alle Empfänger. Diese Anforderung enthält die Paketnummer des zuletzt gesendeten Paketes. Jeder Empfänger bestätigt die Anforderung entweder mit einem ACK-Paket, falls er das letzte Paket empfangen hat, oder mit einem NAK-Paket mit der Nummer des ersten fehlenden Paketes. Mit den bisherigen Mechanismen hat ein Empfänger zwei Chancen, fehlende Pakete zu erkennen. Die erste Möglichkeit besteht darin, Lücken in der Nummerierung der Pakete festzustellen. Dies ist jedoch nicht ausreichend. Legt der Sender für einen längeren Zeitraum eine Sendepause ein, kann der Empfänger nicht feststellen, ob er alle Pakete empfangen hat oder ob Pakete verloren gegangen sind. Dies kann er nur durch den Empfang einer ACK-Anforderung feststellen. Die Absicherung über die ACK-Anforderung hat jedoch den Nachteil, dass ACK-Anforderungen erst nach Überschreiten einer Zeitschranke gesendet werden. Wählt man diese Zeitschranke zu kurz, dann werden unnötig viele ACK-Meldungen erzwungen. Wählt man ihn jedoch zu lang, dann dauert es eventuell eine längere Zeit, bis ein Empfänger das Fehlen eines Paketes bemerkt. Das Dilemma lässt sich teilweise aufheben, wenn man dem Empfänger mitteilt, dass eine Übertragungsfolge kurzzeitig unterbrochen wird. Dies wird durch ein spezielles Paket signalisiert. Hat der Sender alle Pakete in seinem Sendepuffer gesendet, sendet er ein- oder mehrmals ein kurzes Pausepaket mit der Nummer des zuletzt gesendeten Datenpakets. Für einen Empfänger können nun zwei Situationen eintreten: Entweder er erhält ein Pausepaket, dann kann er mit dessen Nummer seine empfangenen Pakete abgleichen. Erhält er für einen kurzen Zeitraum

kein Paket, ohne dass vorher ein Pausepaket empfangen wurde, dann ist die Wahrscheinlichkeit groß, dass Pakete verloren gegangen sind. In diesem Fall schickt er ein NAK-Paket mit der nächsten erwarteten Paketnummer. Mit diesem zusätzlichen Mechanismus kann die Zeitschranke für eine ACK-Anforderung größer eingestellt werden. Mit einem Wert von 10 Millisekunden lassen sich gute Ergebnisse auch bei einer hohen Verlustrate erzielen. Allerdings lassen sich unterschiedliche Verlustraten nur bedingt testen. Für einen grobe Abschätzung wurde in der Implementierung der Übertragungsschicht zufällig ein bestimmter Anteil der Pakete unterdrückt.

Ein weiteres Problem besteht im Erkennen von Situationen, in denen die Sendeapplikation beendet wurde. Ein Empfänger kann durch Untersuchung des Datenflusses nicht ohne Weiteres unterscheiden, ob ein Sender keine Daten sendet oder ob der Sender nicht mehr existiert. Eine Sendeinstanz wird beendet, wenn die sendende Applikation beendet wird oder wenn eine Fehlersituation eintritt. In einer Cluster-Applikation ist es sehr hilfreich, wenn alle Empfänger feststellen können, dass vom Sender keine Daten mehr zu erwarten sind. In diesem Fall kann es sinnvoll sein, dass auch die empfangende Applikation beendet wird. Anderenfalls müsste ein Anwender von Hand die verteilten Programme im Cluster beenden, um einen geordneten Neustart zu gewährleisten. In vielen Protokollen sendet der Sender in regelmäßigen Zeitintervallen eine kurze Mitteilung. Bleibt diese Mitteilung für einen längeren Zeitraum aus, geht der Empfänger davon aus, dass der Sender nicht mehr existiert. Es ist allerdings schwierig, ein gutes Zeitintervall hierfür zu bestimmen. Wird es zu groß gewählt, beendet sich die empfangende Applikation erst lange, nachdem der Sender beendet wurde. Wird der Zeitraum zu kurz gewählt, kann es vorkommen, dass aufgrund einer kurzzeitigen Netzwerkbelastung oder einer hohen CPU-Auslastung das Zeitintervall überschritten wird, ohne dass eine Fehlersituation vorliegt. In verbindungsorientierten Socket-Verbindungen wird das Bestehen der Verbindung durch das Betriebssystem überwacht. Hierbei können die beteiligten Betriebssysteme den Abbau der Verbindung geordnet durchführen. Ein solcher Mechanismus ist in einem Protokoll, das vollständig außerhalb des Betriebssystems realisiert ist, nicht umsetzbar. Wenn das Betriebssystem mit den verbindungsorientierten Sockets einen Mechanismus bereitstellt, um festzustellen, ob eine Seite der Verbindung nicht mehr besteht, dann ist es naheliegend, diesen Mechanismus auch für eine Multicast-Übertragung zu nutzten. Eine Netzwerkverbindung wird üblicherweise in zwei Richtungen genutzt. Ein Rechner in einem Cluster kann an viele andere Rechner per Multicast eine Nachricht verschicken. Es muss aber auch möglich sein, dass der selbe Rechner von allen anderen Rechnern eine Nachricht erhält. In der hier vorgestellten Lösung wird für das Senden ein Protokoll auf der Basis einer paketorientierten Übertragung verwendet. Für das Empfangen von Daten von einzelnen Knoten wird jedoch eine verbindungsorientierte Kommunikation auf der Basis von Stream-Sockets verwendet. Damit entfällt die Aufgabe, über das Multicast-Protokoll den Ausfall des Senders oder eines Empfängers feststellen zu müssen. Der Abbau einer Verbindung wird über den Abbau eines Stream-Socket-Rückkanals angezeigt. Dies ist sehr zuverlässig und ermöglicht es, dass alle Empfänger in einem Zeitraum von weniger als einer Sekunde über das Beenden des Senders informiert werden, unabhängig davon, ob eine sendende Applikation geordnet beendet oder aufgrund einer Fehlersituation abgebrochen wird.

4.4.8 Performanz der entwickelten Protokolle

Das hier vorgestellte verlässliche Multicast-Protokoll wird sowohl mit Gigabit-Ethernet als auch mit dem langsameren 100-Megabit-Fast-Ethernet getestet. Für die Tests stehen unterschiedliche Switches der Firma Cisco zur Verfügung. Für den Test des Protokolls in einem Gigabit-Netzwerk werden zwei Switches der Catalyst-Serie verwendet. Wie bereits im letzten Abschnitt erwähnt, sind diese Switches nicht in der Lage, Pakete per Multicast mit voller Netzwerk-Bandbreite zu übertragen. Aus diesem Grund kann die Leistungsfähigkeit des Protokolls nicht ohne weiteres mit der idealen Bandbreite verglichen werden. Auf dem Catalyst 4003 lassen sich keine Übertragungsraten über 30 Megabyte erreichen. Ein Vergleich mit den Übertragungsraten auf dem etwas leistungsfähigeren Catalyst 4006 zeigt, dass nicht das Protokoll oder die Implementierung, sondern allein die zur Verfügung stehende Hardware für die begrenzte Bandbreite verantwortlich ist.

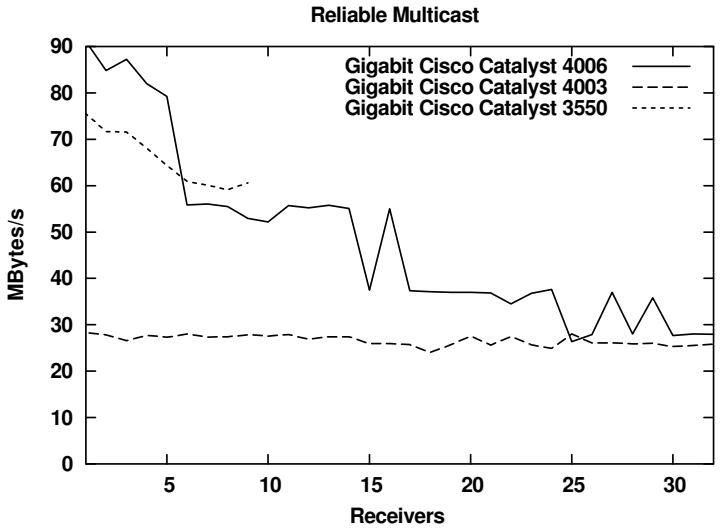

Abbildung 4-9: Datenrate bei Multicast mit Gigabit-Ethernet

Ab etwa 25 Empfängern nähert sich die Übertragungsrate des Catalyst 4006 an die des Catalyst 4003 an. Betrachtet man die Ergebnisse für den Catalyst 4003, so zeigt sich, dass die vom Switch gebotene Bandbreite durch das verlässliche Multicast-Protokoll sehr gut ausgenutzt wird. Es ist auch zu sehen, dass die Anzahl der Empfänger keine Auswirkung auf die Bandbreite hat. Da die Tests aufgrund der für ein schnelles Multicast-Protokoll ungeeigneten Hardware keine eindeutige Aussage über die Leistungsfähigkeit und die Skalierbarkeit des Protokolls liefern, wurde ein weiterer Test in einem 100-Megabit-Ethernet durchgeführt. Die Ergebnisse zeigen, dass die zur Verfügung stehende Bandbreite fast bis zum theoretischen Maximum ausgenutzt wird. Das Protokoll skaliert sehr gut. Bis zu einer Anzahl von 32 Empfängern wird eine gleich hohe Datenrate erreicht.

In einem weiteren Test wurde untersucht, wie die Datenübertragung auf Fehlersituationen reagiert. Während einer Datenübertragung mit 32 Empfängern wurde die Ausführung eines Empfängers beendet. Der Sender wurde im Bruchteil einer Sekunde über dieses Ereignis informiert. Mit einem klassischen Hartbeat-Ansatz wäre dies nicht möglich. Die Robustheit des Protokolls gegenüber

Datenübertragungsfehlern wurde mit dem zufälligen Auslassen der Datenübertragung einzelner Pakete getestet. Diese Tests wurden über einen längeren Zeitraum und mit unterschiedlichen Ausfallwahrscheinlichkeiten durchgeführt. Auch wenn nur sehr wenige Pakete übertragen wurden, blieb die Verbindung bestehen und es wurde ein kontinuierlicher Datenstrom bei den Empfängern ausgeliefert. Allerdings sinkt die Datenrate bei hohen Paketverlusten drastisch. Dies macht sich jedoch erst dann sehr stark bemerkbar, wenn die Ausfallwahrscheinlichkeit 20% übersteigt. Eine solche Fehlerrate ist jedoch in lokalen Netzwerken nicht zu erwarten.

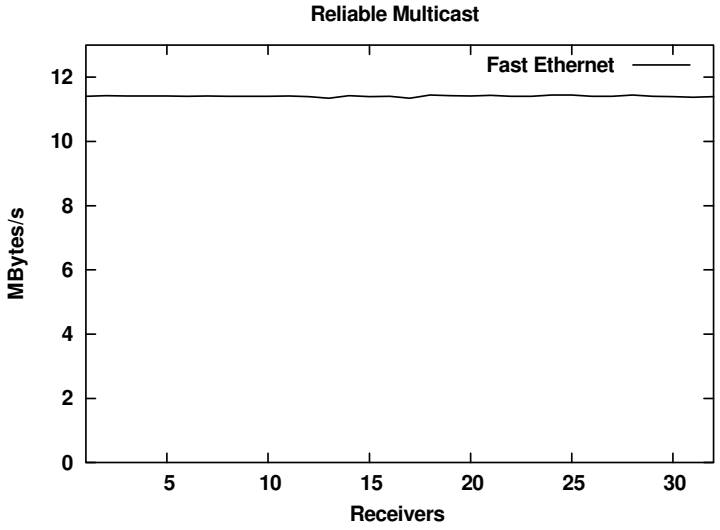

Abbildung 4-10: Reliable Multicast in einem 100-Megabit-Fast-Ethernet

4.4.9 Integration in das Kommunikations-Framework

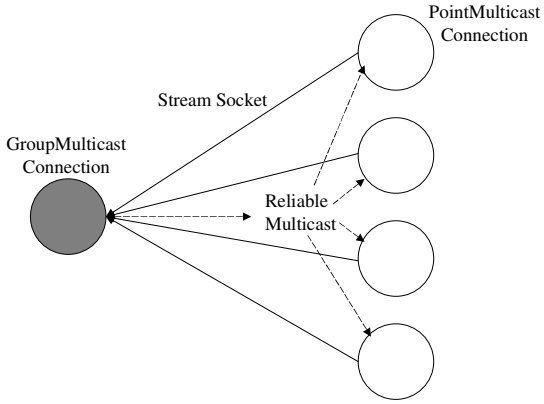

Abbildung 4-11: Multicast Connection

Die Klasse der *MulticastConnection* basiert auf der Klasse der *StreamSocketConnection*. Jede *PointConnection* wird über eine feste Socket-Verbindung an die *GroupConection* angebunden. Die Anbindung der *GroupConnection* an die *PointConnections* erfolgt über das oben beschriebene Multicast-Protokoll. Das Senden und die Protokollverarbeitung werden in einem eigenen Thread

abgearbeitet. Die Anbindung erfolgt über eine FIFO(First In First Out)-Warteschlange. Der Zugriff auf diese Warteschlange wird über Semaphoren und Locks synchronisiert.

4.5 Kommunikation in einer Pipeline

In einem Cluster-System, das mit einer Netzwerkschicht verbunden ist, die kein Broadcast oder Multicast unterstützt, kann die Übertragung großer Datenmengen von einem Rechner des Clusters auf alle anderen Rechner sehr zeitaufwendig sein. Die Daten müssen von der Quelle einzeln an alle anderen Rechner übertragen werden.

Falls das Netzwerk über einen Switch verfügt, der mehreren Verbindungspaaren eine Datenübertragung bei voller Bandbreite erlaubt, lässt sich die fehlende Mehrpunktverbindung durch eine Pipeline ersetzen. Hierbei wird der Datenstrom in kleine Einheiten aufgeteilt. Der Sender sendet das erste Paket an den ersten Rechner der Pipeline. Dieser sendet das Paket an den nachfolgenden Rechner und beginnt mit dem Lesen des folgenden Paketes. Die Pakete werden vom Sender durch die komplette Pipeline übertragen. Je nach Leistungsfähigkeit des Switches können Datenraten erreicht werden, die denen einer Mehrpunktverbindung ähneln.

Der Nachteil der Sende-Pipeline besteht darin, dass bei steigender Anzahl von Rechnern in der Pipeline ein einzelnes Paket immer länger braucht, bis es von allen empfangen wurde. D.h., die Verzögerungszeit hängt linear von der Anzahl der Rechner in der Pipeline ab. Die Datenrate ist demgegenüber unabhängig von der Länge der Pipeline.

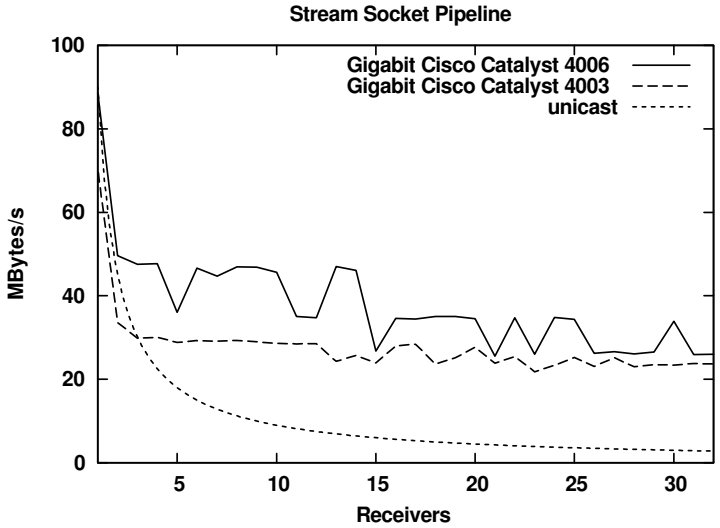

Abbildung 4-12: Datendurchsatz der Socket-Pipeline

Mit der Hilfe einer Kommunikations-Pipeline lassen sich auch schnelle Punkt-zu-Punkt-Verbindungen als Kommunikationsmedium verwenden, die eigentlich nicht zur Vernetzung von Rechnern vorgesehen sind. Beispielsweise lassen sich mehrere Rechner in Reihe per FireWire verbinden. Diese Art der Verbindung ist wesentlich günstiger als Gigabit-Ethernet, da hierbei die Kosten für einen Gigabit-Switch entfallen.

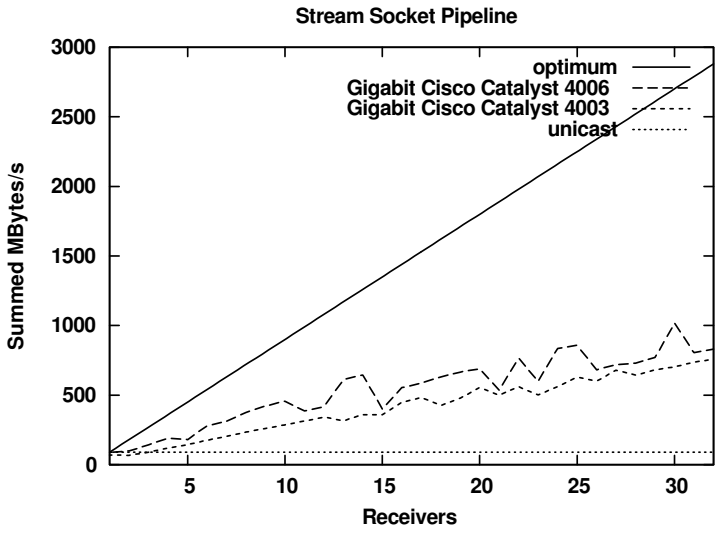

Abbildung 4-13: Summierter Datendurchsatz der Socket-Pipeline

Ein weiteres Einsatzgebiet für Kommunikations-Pipelines sind Netze, die kein Multicast zur Verfügung stellen. Beispielsweise steht in WANs in der Regel kein Multicast zur Verfügung. Mit Hilfe der Kommunikations-Pipeline können auch in solchen Netzen große Datenmengen zwischen vielen Rechnern synchronisiert werden.

Vergleicht man die in Abbildung 4-13 aufgeführten Messergebnisse mit denen des Multicast-Protokolls, so ist die Kommunikation mit Hilfe der Socket-Pipeline nur geringfügig langsamer als das vorher vorgestellte und wesentlich aufwendigere Multicast-Protokoll. Es ist jedoch zu berücksichtigen, dass mit steigender Anzahl an Rechnern in der Pipeline die Zeit immer länger wird, bis ein Paket auf allen Rechnern angekommen ist. Der gemessene Durchsatz gibt nur die maximal zu erwartende Datenrate an, die beim Versenden großer Datenmengen zu erwarten ist.

Aufgrund fehlender Hardware konnten beide Protokolle nicht mit einer größeren Anzahl an Rechnern getestet werden. Es ist jedoch zu erwarten, dass das vorgestellte Multicast-Protokoll auch bei wesentlich größeren Clustern noch akzeptable Reaktionszeiten liefert, während dies bei der Pipeline-Kommunikation nicht mehr der Fall sein dürfte.

4.6 Weitere Anwendungsgebiete

Die oben beschriebenen Mechanismen zur Synchronisation des Szenengraphen und das dabei verwendete Framework ist für die Anforderungen der parallelen Bildberechnung in Cluster-Systemen ausgelegt. Dies schließt jedoch eine Anwendung für andere Aufgaben mit ähnlichen Kommunikationsproblemen nicht aus. Im Folgenden wird beschrieben, wie die entwickelten Techniken für die Realisierung einer Mehrbenutzer-VR-Umgebung verwendet werden können.

In einer VR-Umgebung für mehrere Benutzer werden mehrere VR-Systeme so miteinander verbunden, dass für alle Benutzer der Eindruck einer gemeinsamen einheitlichen Welt entsteht. Die

Darstellungsbasis eines VR-Systems ist der Szenengraph. Bei der Verbindung mehrerer VR-Systeme müssen die einzelnen Szenengraphen miteinander synchronisiert werden. Hierfür können die im Rahmen dieser Arbeit entwickelten Techniken eingesetzt werden.

Bei der parallelen Bildberechnung beschränkt sich die Synchronisation des Szenengraphen darauf, dass eine Client-Applikation alle Änderungen an alle Rechner eines Clusters verteilt. Eine Rücksynchronisation von den Rechnern des Clusters zur Client-Applikaiton ist meist nicht erforderlich. In einer verteilten Multiuser-VR-Umgebung kann jedoch jeder Benutzer Teile des Szenengraphen modifizieren. Diese Modifikationen müssen an alle anderen VR-Systeme gesendet werden.

Für eine Multiuser-Umgebung wird ein zentraler Server vorgesehen. Dieser Server empfängt alle Änderungen von allen VR-Systemen und baut mit diesen Informationen einen Szenengraphen auf, in dem die lokalen Szenengraphen jedes angeschlossenen VR-Systems enthalten sind. Beim Aufbau dieses Master-Szenengraphen entstehen, wie bei jeder Modifikation des Szenengraphen, Änderungslisten. Mit Hilfe dieser Änderungsliste werden alle Änderungen am zentralen Szenengraphen an alle VR-Instanzen gesendet.

Bei dem Versenden des zentralen Szenengraphen ist darauf zu achten, dass jedes einzelne VR-System neue Teile hinzufügen oder löschen kann. Um dies korrekt zu handhaben, wird für jeden Knoten des gemeinsamen Szenengraphens vermerkt, von welchem VR-System er erzeugt wurde. Diese Information ist notwendig, da das Erzeugen eines neuen Knotens im Szenengraphen dazu führt, dass der zentrale Server an alle VR-Systeme die Erzeugung eines neuen Knotens meldet. Das VR-System, das den Knoten eingefügt hat, darf auf eine solche Meldung jedoch nicht noch einmal einen neuen Knoten generieren.

Multiuser-VR-Umgebungen sind ein sehr weites Forschungsgebiet. Mit dem hier vorgestellten Ansatz soll lediglich gezeigt werden, dass es möglich ist, mit den im Rahmen dieser Arbeit entwickelten Techniken ein einfaches Multiuser-System aufzubauen. Es besteht nicht der Anspruch, eine allgemeingültige Lösung für alle Kommunikationsprobleme in Multiuser-Umgebungen zu liefern. Die im Rahmen dieser Arbeit entwickelten Kommunikationsprotokolle sind für den Einsatz in lokalen Netzwerken optimiert, während Multiuser-VR-Umgebungen häufig auch in WANs eingesetzt werden.

4.7 Zusammenfassung

In diesem Kapitel wurde eine Lösung für die schnelle Synchronisation des Szenengraphen in einem Cluster präsentiert. Es wurde ein effizientes und erweiterbares Framework entwickelt, mit dessen Hilfe neben der Übertragung von Daten des Szenengraphen auch beliebige Datenströme wie z.B. Bilddaten übertragen werden können. Zur schnellen Verteilung von Daten an alle Rechner in einem Cluster wurde ein neues Multicast-Protokoll entwickelt. Dieses Protokoll ermöglicht eine zuverlässige Kommunikation und kann auch in sehr großen Cluster-Systemen eingesetzt werden.

Als weiteres Protokoll wurde eine Kommunikations-Pipeline realisiert, mit deren Hilfe in einer Kette aus einzelnen Punkt-zu-Punkt-Verbindungen sehr effizient große Datenmengen übertragen werden können. Das entwickelte Framework und die Synchronisation des Szenengraphen kann auch unabhängig von einer parallelen Bildberechung verwendet werden. In einem eigenen Abschnitt wurde eine Anwendung für eine Multiuser-Umgebung aufgezeigt.

5 Sort-First-parallele Bildberechnung

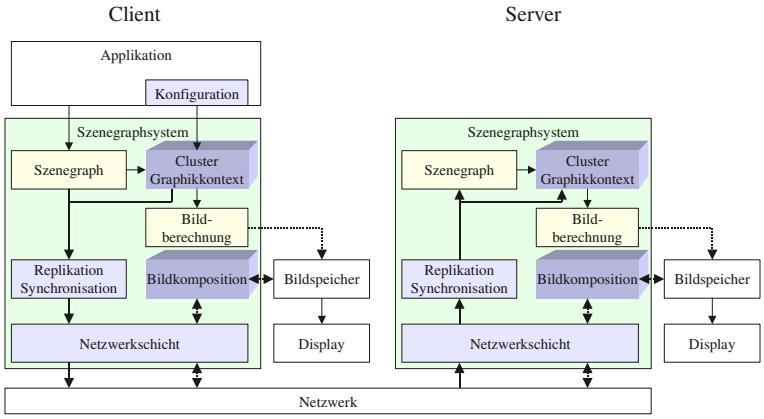

Abbildung 5-1: Sort-First-parallele Bildberechnung

In den vorangegangenen Kapiteln wurde die Verteilung von Szenengraphen und Grafikkontext beschrieben. Auf dieser Basis wird nun die Parallelisierung der Bildberechnung aufgebaut. Die Ansteuerung der parallelen Bildberechung ist Teil des Grafikkontexts. In einem Szenengraphensystem für einen einzelnen Rechner ist der Grafikkontext direkt mit dem Darstellungsbereich auf dem Bildschirm verknüpft. Im Kontext eines Grafik-Clusters wird nicht nur jedem Rechner der lokale Darstellungsbereich zugeordnet, sondern für jeden Rechner im Cluster wird genau festgelegt, welche Aufgabe ihm bei der Bildberechnung zukommt.

Im Folgenden wird die Entwicklung eines Verfahrens für die parallele Bildberechnung in einem Cluster nach dem Sort-First-Prinzip beschrieben. Für die Integration in den Cluster-Szenengraph wird der Grafikkontext erweitert.

Bei der parallelen Bildberechnung nach dem Sort-First-Prinzip wird zuerst der Bildbereich in nicht überlappende Bereiche unterteilt. Anschließend werden die einzelnen Bildausschnitte parallel berechnet. Nach der Berechnung werden alle Bildausschnitte wieder zu einem Gesamtbild zusammengefügt. Wie bereits vorher beschrieben, lässt sich die Bildberechnungszeit mit dem Sort-First-Verfahren nicht beliebig beschleunigen. Wird der Bildbereich in viele kleine Bildausschnitte aufgeteilt, steigt die Anzahl der Polygone an den Bereichsgrenzen, die in mehr als einem Bildausschnitt gezeichnet werden müssen. Ist die Bildberechnungszeit proportional zur Anzahl der Polygone, so kann ab einer gewissen Anzahl von parallel berechneten Bildausschnitten aufgrund der mehrfach berechneten Polygone an den Bereichsgrenzen die Performanz nicht weiter gesteigert werden. Wie viele Bildausschnitte effizient parallel berechnet werden können, hängt sehr stark von der darzustellenden Szene ab. Trotz dieser Einschränkung lässt sich die Berechnungszeit mit einem angemessenen Aufwand je nach Szene durchaus um den Faktor 10 beschleunigen. Das Verfahren erzeugt einen geringen Mehraufwand und ist auch für interaktive Bildwiederholraten geeignet. Da die einzelnen Bildausschnitte ohne Überlappung berechnet werden, sind transparente Objekte und Kantenglättungsalgorithmen der Grafikhardware ohne Einschränkung möglich.

Das Sort-First-Verfahren ist sehr gut für eine dynamische Lastverteilung auf hochauflösenden zusammengesetzten Displays geeignet. Die Darstellung eines Volumendatensatzes oder eines Objektes mit komplexen Materialeigenschaften ist im wesentlichen von der Anzahl der dargestellten Bildpunkte abhängig. Betrachtet man ein solches Objekt auf einem lokalen Bildschirm, so überdeckt das Objekt meist nur einen Bereich in der Mitte des Bildschirms. Betrachtet man nun das gleiche Objekt aus der gleichen Perspektive auf einem gekachelten Display, so ist auf den Kacheln in der Mitte des Displays das Objekt auf der kompletten Bildebene zu sehen, während auf den Kacheln am Rand des Displays häufig nur ein kleiner Randteil des Objektes dargestellt wird. In einer solchen Situation, in der die Berechnungszeit durch die Anzahl der Bildpunkte bestimmt ist, lässt sich mit Hilfe des Sort-First-Verfahrens eine sehr gute Steigerung der Performanz erreichen.

Bisher existieren lediglich Verfahren, die entweder die Berechnung für ein einzelnes Display parallelisieren [SaF98,Mue95,SFL00,CKS02] oder solche, die eine Lastverteilung zwischen den Projektionen eines Multiprojektor-Displays [SFZ99,LCC00] durchführen. Das hier vorgestellte Verfahren erlaubt eine Lastverteilung für beliebige Multiprojektor-Systeme wie z.B. Stereo-Projektion, Cave, Workbench, Stereo- und Mono-Tiled Displays. Das Verfahren ist allgemein verwendbar und kann ohne Änderung beispielsweise auch für die Lastverteilung in einer neuartigen Cave verwendet werden, deren Wände aus gekachelten Einzelprojektionen bestehen.

Bei der Lastverteilung wird untersucht, wie groß der Bildberechnungsaufwand jedes Rechners ist. Es werden hierbei zwei Typen von Rechnern unterschieden. Ein Rechner, an dessen Grafikausgang ein Projektor oder ein Display angeschlossen ist, wird als Display-Server bezeichnet. Ein Rechner, an dem kein Display angeschlossen ist, stellt seine Rechenleistung anderen Rechnern zur Verfügung und wird als Rechen-Server bezeichnet.

Ohne Lastverteilung muss jeder Display-Server den sichtbaren Bereich seiner Projektion selbst berechnen. Im Zuge der Lastverteilung werden nun Bildbereiche neu zugewiesen, so dass im Idealfall jeder Rechner den gleichen Bildberechnungsaufwand besitzt. Dies hat zur Folge, dass die Bildberechnung von der Projektion getrennt werden muss. Bildausschnitte können von beliebigen Rechnern berechnet werden. Um eine korrektes Ergebnisbild zu erhalten, müssen Bildteile über das Netzwerk zwischen den Rechnern ausgetauscht werden.

Für eine optimale Lastverteilung muss ein Maß für den Bildberechnungsaufwand gefunden werden, das als Kriterium für eine Verteilung dient (Lastkriterium). Im Idealfall müsste eine Analyse den exakten Berechnungsaufwand einer Szene ermitteln. Dies ist jedoch mit einem vertretbaren Aufwand nicht möglich. Aus diesem Grund muss eine vereinfachte und damit auch ungenauere Bestimmung des Rechenaufwandes gefunden werden. Die Analyse muss weniger Rechenzeit in Anspruch nehmen als die eigentliche Bildberechnung.

In dieser Arbeit wird zur Bestimmung des Lastkriteriums auf eine Berücksichtigung der Berechnungszeiten aus vorangegangenen Bildberechnungen verzichtet. Theoretisch ließe sich aus

vorherigen Berechnungszeiten sehr genau ein Lastkriterium extrapolieren. Die Extrapolation der Berechnungszeit aus vorangegangenen Berechnungen ist jedoch nur dann genau, wenn sich die Darstellung zwischen dem aktuellen und dem letzten Bild nur wenig unterscheidet. Dies ist dann gegeben, wenn sich die virtuelle Kamera nur wenig bewegt. In einem solchen Fall sind jedoch geringe Bildwiederholraten weniger störend als in den Fällen, in denen sich die virtuelle Kamera schnell bewegt. Schnelle Bewegungen erschweren jedoch die Extrapolation aus vorangegangenen Bildberechnungen.

Die Berechnungszeiten für die darzustellenden Objekte des Szenengraphen lassen sich in einem modernen Szenengraphen nicht eindeutig ermitteln. Dies liegt daran, dass das Szenengraphensystem umfangreiche Optimierungen vornimmt. So wird beispielsweise die Darstellung von Objekten mit gleichen Materialeigenschaften zusammengefasst. Die Berechnungszeit für ein einzelnes Objekt lässt sich nicht mehr eindeutig bestimmen, da sie zu einem großen Teil von vorangegangenen Berechnungen abhängig ist.

Für die Spezifikation des Lastkriteriums werden lediglich Eigenschaften der aktuellen Szene und des aktuellen Betrachtungspunktes berücksichtigt. Dies hat den Vorteil, dass auch bei sehr schnellen Bewegungen der virtuellen Kamera eine gute Lastverteilung möglich ist. Das Verfahren ist auch dann anwendbar, wenn die Szene dynamisch verändert wird. Als Maß für die Berechnung des Lastkriteriums kommen die folgenden Parameter in Frage:

- Anzahl der Bildpunkte im Projektionsbereichs: Dieses Maß lässt sich unabhängig von der darzustellenden Szene bestimmen. Die Anzahl der Bildpunkte ergibt sich allein durch die Auflösung des Displays.

- Anzahl der zu zeichnenden Bildpunkte (Pixel): Dies lässt sich sehr schnell mit Hilfe des Hüllvolumens (Bounding-Box) annäherungsweise bestimmen. Für jedes Objekt des Szenengraphen wird ermittelt, welchen Bildausschnitt dessen Projektion auf dem Bildbereich einnimmt. Die Größe dieser Fläche kann als Maß für die Anzahl der zu zeichnenden Bildpunkte verwendet werden.

- Anzahl der Flächen: Dieses Maß ist etwas problematisch, da Flächen sehr unterschiedlich beschrieben werden können. In einem Szenengraphen werden meist die von OpenGL bekannten Typen *Triangle, Quad, Triangle Stripe, Triangle Fan* und *Polygon* unterstützt. Der Berechnungsaufwand der einzelnen Typen unterscheidet sich sehr stark. Ein genaueres Maß wäre die Anzahl der Dreiecke, die für die Darstellung der einzelnen Typen erforderlich sind. Dieses Maß lässt sich jedoch meist nur durch die Triangulation jeder einzelnen Fläche bestimmen.

- Anzahl der Eckpunkte der dargestellten geometrischen Objekte: Die Anzahl der Eckpunkte lässt sich sehr schnell bestimmen. Viele Eckpunkte sind meist auch ein Indiz für viele Dreiecke.

- Tiefenkomplexität: Dieses Maß gibt an, wie viel Flächen in einem Bildpunkt übereinander gezeichnet werden. Es lässt sich nur schwer exakt bestimmen. Eine Annäherung lässt sich errechnen, wenn man die durchschnittliche Flächengröße, die Flächenzahl und die Größe des Objekts im Bildbereich berücksichtigt.

- Materialeigenschaften der dargestellten Objekte: Die Berechnungszeit für einzelne Objekte des Szenengraphen hängt sehr stark von dessen Materialeigenschaften ab. Materialeigenschaften, die einen starken Einfluss auf die Berechnungszeit haben, sind beispielsweise die Anzahl der Texturen oder Pixel- und Vertex-Shader.

Eine Abschätzung des Berechnungsaufwandes ist auch aus einer Kombination der aufgeführten Maßeinheiten möglich. Bevor in den folgenden Abschnitten genauer auf die Lastermittlung und Lastverteilung eingegangen wird, soll zuerst das Potential von Sort-First-basierten parallelen Bildberechnungsverfahren untersucht werden. Wie bereits erwähnt, sind der Performanzsteigerung durch eine parallele Berechnung des Bildbereiches Grenzen gesetzt. Bisher wurde diese Grenze beispielsweise von Eyles und Molnar [MCE94] nur analytisch für den Fall einer gleichmäßigen Verteilung von Polygonen bestimmt. Im folgenden Abschnitt soll für konkrete Szenen die maximale Performanzsteigerung ermittelt werden, die durch eine parallele Berechnung von Bildteilen erzielbar ist.

5.1 Das Potential von Sort-First

In diesem Abschnitt soll das Potential des Sort-First-Verfahrens untersucht werden. Dazu wird eine obere Schranke für die maximal erreichbare Bildwiederholrate für eine Szene bei einer gegebene Grafikhardware bestimmt. Sort-First-basierte Verfahren lassen sich nicht beliebig skalieren, da Polygone, die in mehreren parallel berechneten Bildausschnitten zu sehen sind, auch mehrfach gezeichnet werden müssen. Die Anzahl der mehrfach gezeichneten Polygone ist als Maß nicht ausreichend, um für eine gegebene Szene eine optimale Lastverteilung zu bestimmen. Dies liegt daran, dass die Berechnungszeit von Eigenschaften einzelner Polygone abhängig ist, von deren Gruppierung in einem Szenengraph, von Materialeigenschaften wie z.B. Texturen und Shadern, von Optimierungsstrategien der Grafikhardware und des Szenengraphensystems und auch von der Aufteilung des Bildbereichs.

Die obere Schranke für die maximal zu erreichende Bildwiederholrate ist hilfreich, wenn es um die Bewertung von Verfahren zur Lastverteilung geht. Sie kann auch dazu verwendet werden, die Auswirkung von Optimierungen der Szenen auf die Effizienz von Sort-First eindeutig zu bestimmen.

Im Folgenden wird deshalb ein Verfahren entwickelt, mit dessen Hilfe die obere Schranke für die maximal zu erreichende Bildwiederholrate ermittelt werden kann. Eine optimale Lastverteilung muss den Bildbereich so aufteilen, dass bei einer gegebenen Anzahl an Rechnern die

Bildberechnung optimal parallelisiert werden kann. Diese Aufgabe lässt sich jedoch nicht in einer vertretbaren Zeit exakt lösen. Für die Bestimmung der oberen Schranke ist dies allerdings nicht erforderlich. Hier wird davon ausgegangen, dass die Anzahl der parallel arbeitenden Rechner nicht beschränkt ist. In diesem Fall kann jeder Bildpunkt einem Rechner zugewiesen werden. Die erreichbare Bildwiederholrate ergibt sich dann aus dem maximalen Berechnungsaufwand für einen einzelnen Bildpunkt. Da die Berechnung für einen Bildpunkt auf der Basis von Sort-First nicht weiter parallelisiert werden kann, kann die Bildberechnung auch nicht durch den Einsatz weiterer Rechner beschleunigt werden.

Für die Ermittlung der oberen Schranke muss also der Bildpunkt gefunden werden, dessen Berechnung die meiste Rechenzeit erfordert. Passt man die Projektion so an, dass jeweils nur ein Bildpunkt überdeckt ist, so kann man nacheinander die Berechnungszeiten für jeden Bildpunkt ermitteln. Dies kann bei komplexen Szenen und hohen Bildschirmauflösungen jedoch schnell einige Tage an Rechenzeit in Anspruch nehmen. Aus diesem Grund wird auf der Basis einer Intervallschachtelung ein schnellerer Ansatz entworfen.

```
rect = whole window
t = display(szene,rect)
saveRect(t,rect)
do
       (rect1,rect2) = split(rect)
       t = display(szene,rect1)
       saveRect(t,rect1)
       t = display(szene,rect2)
       saveRect(t,rect2)
       (t,rect) = findMaxT()
while(pixelSize(rect) gt 1)
print "Maximale Bildwiederholrate",1/t
```

Der hier vorgestellte Algorithmus unterteilt den Bildausschnitt, der die meiste Rechenzeit erfordert, in zwei kleinere Bereiche. Dies wird so lange wiederholt, bis der aufwendigste Bildausschnitt nur noch einen einzelnen Bildpunkt überdeckt. Mit diesem Ansatz müssen nicht mehr alle Bildpunkte einzeln berechnet werden, sondern das Maximum wird bereits nach wenigen hundert Iterationsschritten bestimmt.

Szene	Polygone	Fps single	Resolution	Fps	Areas
Power Plant	13 Mil.	1,6	320 x 200	22	187
			640 x 480	25	338
			1024 x 768	29	673
Stanford Dragon	8 Mil.	2,8	320 x 200	159	245
			640 x 480	182	339
			1024 x 768	187	324
Stanford Lucy	20 Mil.	1,1	320 x 200	9,8	269
			640 x 480	10,3	442

BMW	4 Mil.	3,2	1024 x 768	10,4	593
			320 x 200	187	196
			800 x 600	200	237
			1024 x 768	201	246

Tabelle 5-1: Maximale Bildwiederholrate (NVidia Quadro FX 1100, Pentium 2.4GZ)

Die hier aufgeführten Zahlen sind abhängig von der Position der virtuellen Kamera, von der Auflösung des Displays und der eingesetzten Hardware. Sie sind ein sehr gutes Maß dafür, welche Performanzsteigerung für eine Szene maximal erreichbar ist. Mit einem Algorithmus zur dynamischen Lastverteilung sind diese Werte nicht zu erreichen. In der Regel ist die Anzahl der verfügbaren Rechner beschränkt, und die Lastverteilung bleibt weit unter der berechneten oberen Schranke zurück.

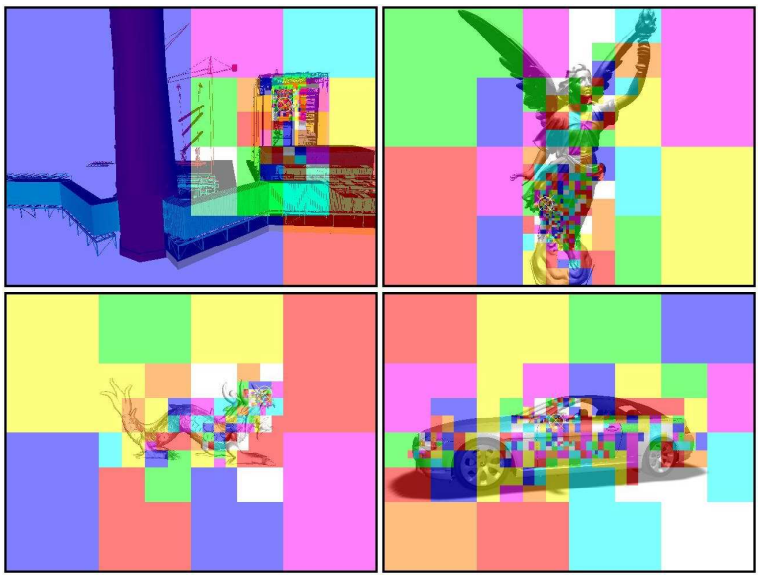

Abbildung 5-2: Limitierender Bildbereich für Sort-First

Abbildung 5-2 zeigt für die in Tabelle 5-1 untersuchten Szenen die Bereiche, die aufgrund ihrer Komplexität die Steigerung der Bildwiederholrate mittels Sort-First beschränken. Die Position des Bildpunktes mit der größten Komplexität ist mit einem Kreis markiert.

Aus den ermittelten Werten lassen sich folgende Schlussfolgerungen ziehen:

- Die maximale Bildwiederholrate ist nicht durch die Anzahl der eingesetzten parallel arbeitenden Rechner, sondern durch die Komplexität und Struktur der Szene beschränkt.

- In den meisten Szenen existieren Bereiche mit hoher und Bereiche mit geringer Komplexität. Eine effiziente dynamische Lastverteilung muss deshalb sowohl den Betrachtungspunkt als auch die Komplexitätsverteilung innerhalb der Szene berücksichtigen.

- Das Potential des Sort-First-Verfahrens steigt mit der Auflösung des Displays. Der Sort-First-Ansatz ist daher sehr gut für hochauflösende zusammengesetzte Displays geeignet.

Im Folgenden wird ein Verfahren vorgestellt, mit dessen Hilfe es möglich ist, den Bildberechnungsaufwand in einem Cluster zu verteilen. Das Verfahren ist in der Lage, die Bildberechnung eines einzelnen Bildes auf mehrere Rechner zu verteilen und bei einem Projektionssystem mit mehreren Projektoren die Last zwischen den einzelnen Darstellungs-Servern gleichmäßig aufzuteilen. Es ist möglich, für jedes Projektionssystem beliebig viele zusätzliche Rechner zur Beschleunigung einzusetzen.

5.2 Lastermittlung

Bei fast allen existierenden Verfahren zur Lastermittlung für Sort-First wird der Bildbereich in ein zweidimensionales Gitter aufgeteilt. Anschließend wird für jede Gitterzelle die Berechnungszeit geschätzt. Eine Übersicht weit verbreiteter Verfahren ist beispielsweise in [Mue95] zu finden. Meist wird für jede Gitterzelle ermittelt, wie viele Polygone in der Zelle sichtbar sind. Die Lastverteilung bezieht sich ausschließlich auf die ermittelte Polygonzahl pro Gitterpunkt.

Dieser Ansatz führt jedoch in einem modernen szenengraph-basierten System zu einer ungenauen Lastabschätzung. Der Grund hierfür liegt darin, dass bei dieser Art der Lastermittlung das Wissen über den Objektzusammenhang verloren geht. In einem Szenengraphensystem werden viele Polygone zu grafischen Objekten zusammengefasst und als ganzes an die Grafikhardware übergeben. Die anfallenden Kosten für die Bildberechnung können nur zu einem Teil einer Gitterzelle zugeordnet werden. Für eine Gitterzelle können nur die Berechnungskosten bestimmt werden, die mit der direkten Darstellung von Polygonen zusammenhängen. Hierzu zählen vor allem die Kosten für das Setzen von Bildpunkten. Daneben fallen jedoch auch Kosten an, die nicht einer einzelnen Gitterzelle zugeordneten werden können. Dazu zählen Vertex-Shader und die komplette Transformationsberechnung. Für ein Objekt, das vom Szenengraphensystem an die Grafikhardware übergeben wird, müssen alle Eckpunkte der Geometrie von der Hardware in den Bildraum transformiert werden. Diese Kosten können nicht für einzelne Gitterzellen des Bildschirms berechnet werden. Eine Berücksichtigung dieser Lasten in der Verteilung ist nur dann möglich, wenn die Strukturierung des Szenengraphen berücksichtigt wird. Aus diesem Grund wird hier ein neuer Ansatz für die Lastermittlung und Lastverteilung entworfen, der die szenengraph-spezifischen Lasten berücksichtigt und der somit alle anfallenden Kosten korrekt erfasst.

Dieser Algorithmus zur Lastverteilung setzt auf einem Szenengraphensystem auf. Die Blätter des Szenengraphen enthalten die darstellbare Geometrie. Ein Geometrieobjekt kann aus sehr vielen einzelnen Polygonen zusammengesetzt sein. Meist besitzen alle Polygone einer Geometrie die gleichen Materialeigenschaften und können daher sehr effizient von der Grafikhardware als ganzes verarbeitet werden. Das Szenengraphensystem prüft für jedes Blatt des Graphen, ob die darin enthaltene Geometrie im Bildbereich sichtbar ist. Ist ein Geometrieobjekt vollständig oder teilweise sichtbar, wird das entsprechende Blatt des Szenengraphen als ganzes an die Grafikhardware

übergeben. Der Sichtbarkeitstest wird auch als Culling bezeichnet und ist meist sehr effizient umgesetzt. Hierbei wird, zur Steigerung der Performanz, jedoch lediglich die Hüllgeometrie eines Objektes berücksichtigt.

Für die Lastermittlung müssen daher zwei Fälle unterschieden werden. Ein Objekt, das nicht sichtbar ist, verursacht keine Kosten. Der Aufwand für den Sichtbarkeitstest des Szenengraphensystems ist so gering, dass er hier vernachlässigt werden kann. Ein teilweise oder vollständig sichtbares Objekt muss von der Grafikhardware bearbeitet werden. Hierfür ist eine Kostenabschätzung erforderlich. Um eine möglichst realistische Kostenabschätzung zu erhalten, werden bei der Lastermittlung die folgenden drei Kostenarten betrachtet:

1. **Konstante Kosten C_k**: Diese Kosten fallen für alle Objekte an, die an die Grafikhardware gesendet werden. Sie sind unabhängig davon, ob das Objekt vollständig oder nur teilweise sichtbar ist. Im wesentlichen wird damit der Rechenaufwand für die Transformation abgedeckt. Diese Kosten sind proportional zur Anzahl der indizierten Polygoneckpunkte eines Objektes, welche sehr schnell ermittelt werden können.

2. **Relative Kosten C_r**: Diese Kosten sind davon abhängig, welcher Anteil eines Objektes sichtbar ist. Die relativen Kosten sind jedoch unabhängig von der Darstellungsgröße. Für die Abschätzung dieser Kosten wird für jedes Objekt das Rechteck bestimmt, das das Objekt in der gesamten Bildebene (unabhängig vom Bildschirmfenster) überdeckt (Bildebenenrechteck r). Ebenso wird das Rechteck berechnet, das den Anteil des Objektes im darzustellenden Bildausschnitt überdeckt (Bildausschnittsrechteck a). Als Abschätzung für die sichtabhängigen Kosten werden diese beiden Rechtecke in Bezug zueinander gesetzt. Der Flächeninhalt des Bildausschnittrechtecks vom Bildebenenrechteck wird mit der Anzahl der indizierten multipliziert.

3. **Pixel-Kosten C_p**: Diese Kosten fallen für jeden sichtbaren Bildpunkt eines Objektes an. Sie werden mit der Fläche des Bildausschnittsrechtecks abgeschätzt. Diese Kosten sind sehr stark von den Materialeigenschaften eines Objektes abhängig.

Die Berücksichtigung von konstanten, relativen und Pixel-Kosten ermöglicht eine sehr effiziente Lastverteilung. Ist beispielsweise ein Objekt in mehreren Bildausschnitten sichtbar, die parallel berechnet werden, so fallen die konstanten Kosten mehrfach an. Dies ist der Grund für die beschränkte Parallelisierbarkeit des Sort-First-Verfahrens. Da die konstanten Kosten in dem hier vorgestellten Verfahren in der Kostenfunktion berücksichtigt werden, kann die Lastverteilung diese berücksichtigen.

$$c(o) = i(o)\left(C_k + C_r \frac{a(o)}{r(o)} \right) + C_p a(o)$$

Für die Lastermittlung werden Abschätzungen verwendet, die sehr schnell ermittelt werden können. Der aufwendigste Teil der Berechnung besteht darin, das Bildebenenrechteck – also das umschließende Rechteck in der Bildebene – zu bestimmen. Hierzu müssten theoretisch alle Eckpunkte eines Objektes des Szenengraphen in die Bildebene projiziert werden. Szenengraphensysteme beinhalten jedoch für jedes Objekt ein Hüllvolumen (Bounding Volume), das das Objekt im Objektraum umschließt. Für eine Abschätzung des umschließenden Rechtecks ist es daher ausreichend, das Bounding Volume in die Bildebene zu projizieren. Wird vom Szenengraphensystem ein Quader zur Definition des Bounding Volume verwendet, müssen lediglich die acht Eckpunkte des Quaders projiziert werden. Aus den acht projizierten Punkten werden die minimalen und maximalen Werte ermittelt, die dann das umschließende Rechteck ergeben.

Die Lastermittlung für Szenen mit sehr vielen Objekten kann mit der hier vorgestellten Abschätzung sehr zeitaufwendig sein. Um auch in solchen Szenen eine effiziente Lastermittlung durchführen zu können, werden mehrere Objekte des Szenengraphen zusammengefasst und bei der Lastermittlung und Verteilung als ein einzelnes Objekt betrachtet. Dabei gehen zwar Informationen über die einzelnen Objekte verloren, es wird jedoch ein Instrumentarium geschaffen, um den Rechenaufwand für die Lastermittlung zu verringern. Dies ist insbesondere dann wichtig, wenn bei der Darstellung feste oder minimale Bildwiederholraten erforderlich sind.

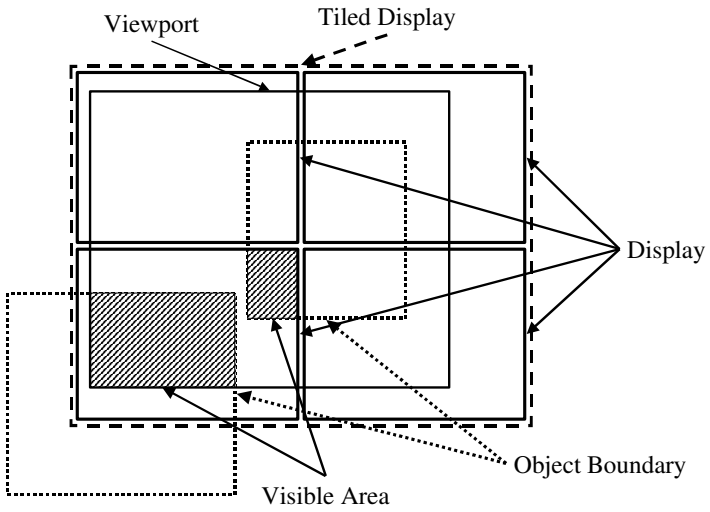

Abbildung 5-3: Lastermittlung für ein Tiled Display

Wie bereits oben erwähnt, soll der in dieser Arbeit entwickelte Algorithmus zur dynamischen Lastverteilung nicht nur für die parallele Berechnung eines Bildes für ein einzelnes Display, sondern auch für hochauflösende zusammengesetzte Projektionssysteme eingesetzt werden. Abbildung 5-3 zeigt den schematischen Aufbau eines Tiled Displays mit vier Projektoren. Die Ansteuerung eines solchen Displays soll für die Anwendung unsichtbar sein. Das bedeutet, dass eine Anwendung ein Tiled Display in gleicher Weise verwenden kann wie ein einzelnes Display. Eine Applikation kann auf einem Display beliebig viele unabhängige Bildausschnitte definieren, in denen Objekte aus einer bestimmten Position der virtuellen Kamera dargestellt werden. Diese werden als Viewports bezeichnet. Auf einem Tiled Display können sich diese Viewports über

mehrere Projektionsflächen erstrecken. Innerhalb eines Viewports werden vom Szenengraphensystem die grafischen Objekte gezeichnet.

Die Lastermittlung muss für jedes Display ermitteln, welche Viewports sichtbar sind. Anschließend muss die Ausdehnung aller Objekte des Szenengraphen innerhalb eines Viewports ermittelt werden. Aus diesen Informationen kann dann die Last aller sichtbaren Objekte abgeschätzt werden. Da in jedem Viewport eine vollständig andere Szene dargestellt werden kann, muss bei der Lastermittlung eine hierarchische Struktur aufgebaut werden. Ein Server betreibt ein Display und ist für einen Teilbereich des Tiled Displays zuständig. In diesem Bereich sind verschiedene Viewports teilweise sichtbar. In einem Viewport müssen Objekte des Szenengraphen dargestellt werden. Ein Objekt ist auf einer Fläche innerhalb des Viewports sichtbar. Ausgehend von dieser Struktur werden für jedes Objekt die konstanten, relativen und pixelbasierten Kosten geschätzt.

Die Lastermittlung muss für jedes Display getrennt durchgeführt werden. Da die Lastermittlung der einzelnen Displays unabhängig voneinander ist, kann sie parallel auf den Display-Servern durchgeführt werden. Erst für die nachfolgende Lastverteilung müssen alle Einzelergebnisse zusammengeführt werden. Mit dieser Parallelisierung kann auch für Tiled Displays mit sehr vielen Einzelprojektionen sehr schnell eine Lastermittlung durchgeführt werden.

Alle parallel ermittelten Ergebnisse der Lastermittlung werden an einen zentralen Rechner gesendet. Der folgende Abschnitt beschreibt, wie aufgrund dieser Informationen die Last zwischen allen beteiligten Rechnern aufgeteilt wird.

5.3 Vorüberlegungen zur Lastverteilung

Beim Entwurf eines Systems zur dynamischen Lastverteilung ist es sehr wichtig, dem System das „richtige“ Ziel vorzugeben. Im Falle von Sort-First könnte man das Ziel beispielsweise so formulieren: „Der Berechnungsaufwand zur Bilddarstellung für alle Bildausschnitte soll gleich sein.“ Auf den ersten Blick scheint damit eine durchaus gute Zielsetzung definiert zu sein. Im Falle von Sort-First muss eine gleichmäßige Verteilung der Rechenlast jedoch nicht unbedingt eine optimale Lastverteilung mit sich bringen. Dies soll an einem Beispiel verdeutlicht werden.

Auf einem Bildschirm sind zwei Polygone sichtbar. Nun soll der Bildbereich in zwei Teile mit gleicher Berechnungszeit aufgeteilt werden. Werden die Bereiche so gelegt, dass jeweils ein Polygon in jedem Bereich sichtbar ist, so beträgt die Anzahl der zu bearbeitenden Polygone zwei. Werden die Bereiche jedoch so gelegt, dass jeder Bereich die Hälfte beider Polygone überdeckt, dann sind in jedem Teilbereich zwei Polygone sichtbar und es müssen in der Summe vier Polygone bearbeitet werden. Dies macht deutlich, dass das Ziel einer ausgewogenen Berechnungszeit im Falle von Sort-First nicht zum optimalen Ergebnis führen muss.

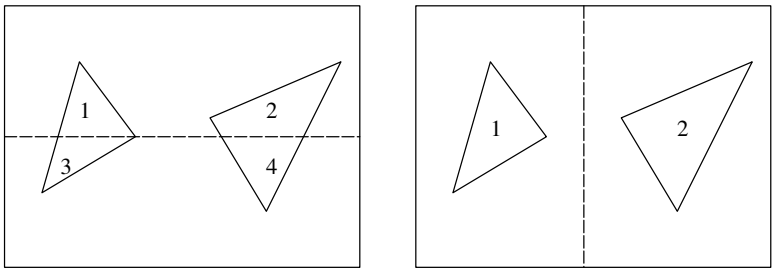

Abbildung 5-4: Polygon-Anzahl bei unterschiedlichen Bildschirmaufteilungen

Aus diesem Grund wird in dem hier neu entwickelten Verfahren die Aufgabe der Lastverteilung wie folgt formuliert: „Suche eine Aufteilung der Bildausschnitte, die eine möglichst hohe Bildwiederholrate des gesamten Bildbereiches bewirkt." Bei einer parallelen Bildberechnung für verschiedene Bildausschnitte wird die Berechnungszeit durch die aufwendigste Berechnung bestimmt. Bei der Bewertung von alternativen Bildaufteilungen müssen daher nur die jeweils rechenintensivsten Bereiche verglichen werden.

Auf der Basis der oben beschriebenen Lastermittlung würde das Beispiel in Abbildung 5-4 für beide alternativen Schnitte die folgenden Kosten ergeben. Zur Vereinfachung wird angenommen, dass beim waagerechten Schnitt beide Polygone jeweils in zwei gleiche Teile zerschnitten werden.

Schnitt 1 (waagerecht)

$$K_1 = 2\left(C_k + \frac{C_r}{2} + C_p \frac{A}{2} \right)$$

Schnitt 2 (senkrecht)

$$K_2 = C_k + C_r + C_p \cdot A$$

Betrachtet man beide Ergebnisse, so sind die relativen und die pixelbasierten Kosten bei beiden Lösungen identisch. Lediglich der konstante Anteil in der Kostenfunktion erlaubt eine Bevorzugung des günstigeren waagerechten Schnittes.

Üblicherweise wird bei der Lastverteilung für Sort-First ein Bereich so aufgeteilt, dass die längere Seite des Bildschirmausschnitts geteilt wird. Dies wird auch als „median cut" bezeichnet [Whe 85,Mue95]. Der Umfang der resultierenden Bereiche wird durch den „median cut" minimiert und somit auch der Überlappungsfaktor [MCE94]. Da das hier beschriebene Verfahren bei der Kostenbestimmung bereits die überlappenden Teile der Geometrie berücksichtigt, kann auf eine sukzessive Unterteilung entlang der X- und Y-Achse verzichtet werden. Der Überlappungsfaktor gibt nur eine statistische Wahrscheinlichkeit für eine Überlappung mehrerer Bereiche an. Abbildung 5-5 zeigt, dass der Schnitt der längsten Kante durchaus zu schlechteren Ergebnissen führen kann. Im Bild auf der linken Seite muss jeder Rechner zwei Objekte bearbeiten. Das hier vorgestellte Verfahren führt zu dem auf der rechten Seite gezeigten Ergebnis. Hier muss jeder Rechner nur ein Objekt bearbeiten.

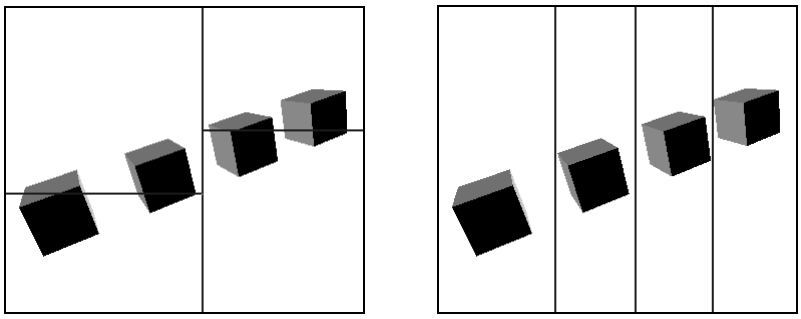

Abbildung 5-5: Bildunterteilung nach "median cut" und "best cut"

Als Ziel der dynamischen Bildunterteilung wird die Minimierung der resultierenden Berechnungszeit angestrebt, d.h., es kann durchaus ein Teilbereich weniger Rechenaufwand erfordern als ein anderer. Es wird lediglich angestrebt, dass es für den rechenintensivsten Teilbereich keine bessere Alternative gibt.

5.4 Lastverteilung

Wie oben beschrieben, liefert die Lastermittlung eine Lastabschätzung für alle beteiligten Rechner. Die Last ist dabei so aufgeschlüsselt, dass Aussagen über jedes Objekt des Szenengraphen bei unterschiedlichten Aufteilungen des Bildbereiches gemacht werden können. In einem ersten Schritt werden alle Server mit ihrer Rechenlast in einer Tabelle gespeichert und absteigend nach dem Berechnungsaufwand sortiert.

Server	Last
1	900
2	500
3	190
4	10

Tabelle 5-2: Lasttabelle

Anschließend wird aus allen Lasten eine durchschnittliche Last ermittelt. Im Beispiel in Tabelle 5-2 beträgt die durchschnittliche Last 400. Die Tabelle wird nun so verarbeitet, dass jeweils der Rechner mit der höchsten Last dem Rechner mit der geringsten Last so viel Rechenzeit aufbürdet, bis alle beteiligten Rechner eine durchschnittliche Last zu bewältigen haben.

Server	Eigene Last	Fremde Last	Für Server
1	400		
2	400		
3	190	110	1
		100	2
4	10	390	1

Tabelle 5-3: Lastverteilung zwischen den Servern

Bei dieser Art der Verteilung werden nur die globalen Kosten für jeden Server betrachtet. Das Ergebnis dieses Arbeitsschrittes bildet die Grundlage für die Suche nach einer optimalen Aufteilung des Bildbereiches.

Die Suche nach einer optimalen Aufteilung lässt sich sehr effizient für die Unterteilung in zwei Bereiche umsetzen. Der hier verfolgte Ansatz basiert auf dem in [SZF99] vorgestellten KD-Split-Algorithmus. Zuerst werden für einen zu unterteilenden Bildbereich alle sichtbaren Objekte ermittelt. Entlang einer Achse des Bereichs werden nun für jeden Bildpunkt die Objekte in einer Tabelle gespeichert, die an diesem Bildpunkt beginnen, und in einer anderen Tabelle werden die Objekte gespeichert, die an dem gleichen Bildpunkt enden.

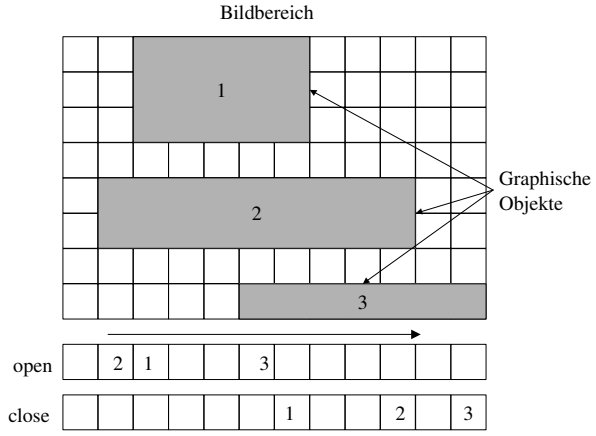

Abbildung 5-6: Unterteilung eines Bildbereiches

Die Tabellen der öffnenden und schließenden Objekte können ohne großen Rechenaufwand aufgebaut werden. Auf der Basis dieser beiden Tabellen kann nun ein effizienter Aufteilungsalgorithmus umgesetzt werden. Dieser Algorithmus sucht entlang einer Koordinatenachse den Schnitt, der die geringsten maximalen Kosten verursacht.

```
cLeft = areaCost
cRight = 0
r = 0
for cut = beginArea to endArea
      foreach o in openList[cut]
            r += o.relative + o.pixel
           cLeft += o.fix
       cLeft += r;
       cRight -= r;
       foreach o in closeList[cut]
            r -= o.relative + o.pixel
            cRight -= o.fix
       maxC = max(cLeft,cRight)
       If(cLeft ge cRight)
            break
       lastMaxC = maxC
       lastCLeft = cLeft
       lastCRight = cRight
If(lastMaxC lt maxC)
```

```
            return (cut-1,lastCLeft,lastCRight)
    else
            return (cut,cLeft,cRigth)
```

Mit dem hier aufgezeigten Algorithmus existiert eine sehr effiziente Möglichkeit, einen Bildbereich in zwei Bereiche mit jeweils gleicher Last zu unterteilen. Der Algorithmus kann in dieser Form jedoch nicht für die Unterteilung in mehr als zwei Bereiche oder in Bereiche unterschiedlicher Last verwendet werden.

Um einen Bereich in mehrere Unterbereiche zu unterteilen, wird der Algorithmus mehrfach rekursiv aufgerufen. Soll ein Bereich beispielsweise auf drei Server verteilt werden, so wird er zuerst in einem Verhältnis von 2 zu 1 in zwei Bereiche unterteilt. Anschließend wird der größere Bereich noch einmal im Verhältnis 1 zu 1 unterteilt. Hierzu muss jedoch eine Unterteilung mit einem vorgegebenen Verhältnis möglich sein. Um dies zu ermöglichen, werden die Kosten c1 und c2 im oben aufgeführten Algorithmus jeweils mit einem Faktor multipliziert, der das gewünschte Verhältnis bestimmt.

Der Algorithmus ist in der Lage, auf der Basis der Kostenabschätzung einen optimalen Schnitt entlang einer Koordinatenachse zu bestimmen. Jeder Bereich kann jedoch entlang beider Bildschirmachsen unterteilt werden. Um ein optimales Ergebnis zu erzielen, werden für jeden Schnitt beide Achsen betrachtet, und anschließend wird der Schnitt mit den geringsten maximalen Kosten beider Seiten ausgewählt.

Im Beispiel aus Tabelle 5-3 wird der Bildbereich des Servers 1 im Verhältnis 400, 110 und 390 auf die Server 1, 3 und 4 aufgeteilt. Der Bildbereich des Servers 2 wird im Verhältnis 400 zu 100 auf die Server 2 und 3 verteilt.

Das Ergebnis der Lastverteilung ist eine Liste von Bildausschnitten, durch die eine Verteilung der Bildberechnung in verschiedenen Arbeitspaketen auf die einzelnen Rechner zugewiesen wird. Die Arbeitspakete werden über das Netzwerk verteilt und anschließend parallel bearbeitet.

Der wesentliche Unterschied dieses Verfahrens zum KD-Split-Algorithmus in [SFZ99] besteht darin, dass dort der Bildbereich in n Teile gleicher Last unterteilt wird. Bei einem Tiled Display kann dies dazu führen, dass einzelne Bildbereiche größer sind als der lokale Bildspeicher der Grafikhardware. Diese Bereiche müssen unterteilt und in mehreren Schritten berechnet werden. Dabei entstehen jedoch wieder Kosten für das mehrfache Zeichnen einzelner Objekte. Der hier verfolgte Ansatz geht davon aus, dass jedem Rechner ein Teil des Bildbereiches zugeordnet wird. Ausgehend von dieser Zuordnung wird eine Lastabschätzung durchgeführt. Anschließend werden einzelnen Servern mit zu hoher Last Bildbereiche abgenommen und von Servern mit geringer Last berechnet. Dieser Ansatz hat den Vorteil, dass er auch dann verwendbar ist, wenn das zusammengesetzte Projektionssystem keine homogene Fläche bildet.

5.5 Weitere Optimierungen

Unter der Voraussetzung, dass die Lastermittlung realistische Werte für die Bildberechnung ermittelt, liefert die Lastverteilung bereits gute Ergebnisse. Weitere Optimierungen sind jedoch möglich, wenn berücksichtigt wird, dass die parallel berechneten Bildausschnitte wieder über ein Netzwerk versendet werden müssen, um ein korrektes Endergebnis zu erhalten. Weitere umgesetzte Optimierungsstrategien sind:

- Wird ein Bildbereich in mehrere Bildausschnitte unterteilt, dann wird dem Server, der diesen Bildausschnitt darstellen soll, der flächenmäßig größte Anteil zugewiesen. Damit lassen sich die zu übertragenden Bilddaten bei einer ungleichen Komplexitätsverteilung im Bildraum erheblich reduzieren.

- Es werden nur dann Arbeitspakete an andere Rechner übergeben, wenn die Lastverminderung den Aufwand der Bildübertragung übersteigt.

5.6 Übertragung der Bilddaten

Bei allen parallelen Sort-First-Verfahren müssen nach der Bildberechnung Bildteile über ein Netzwerk zwischen den Rechnern ausgetauscht werden. Hierfür kann Spezialhardware wie z.B. Sepia [MHS99], Metabuffer [BBF0] oder Lightning-2 [SEP01] verwendet werden. Meist werden dabei die Bilddaten direkt über den digitalen Bildausgang der Grafikkarte ausgelesen. Es treten nur sehr geringe Verzögerungszeiten bei der Zusammensetzung des Ergebnisbildes auf. Der Hardwareansatz hat jedoch auch Nachteile gegenüber einer Übertragung über ein Standard-Netzwerk. Spezialhardware ist teuer und bietet meist nur eine beschränkte Anzahl von Ein- und Ausgängen. Mit Hilfe von Ethernet- oder Myrinet-Netzwerken lassen sich dagegen beliebig viele Rechner parallel betreiben.

5.6.1 Ablauf der Bildkomposition

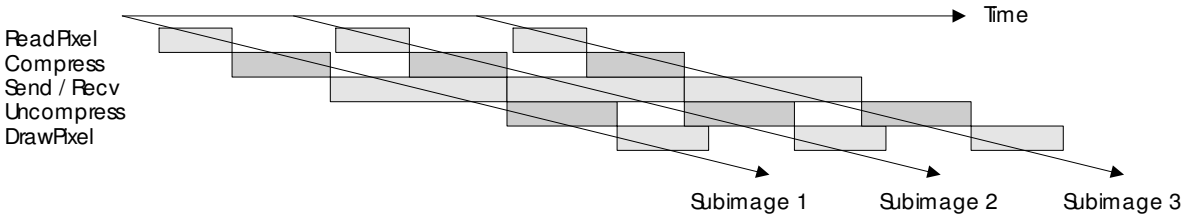

Abbildung 5-7: Parallele Pipeline zur Bildübertragung

Um die Übertragung über das Netzwerk möglichst effizient durchzuführen, wird versucht, alle beteiligten Komponenten möglichst gleichmäßig auszulasten. In einem modernen PC können Grafikhardware, CPU und Netzwerk parallel betrieben werden. Da oft gemeinsame Datenpfade verwendet werden, arbeiten die Einheiten nicht vollständig unabhängig voneinander. Das

Versenden der Bilddaten lässt sich dennoch beschleunigen, wenn versucht wird, alle Einheiten gleichzeitig zu betreiben. Bei der Übertragung müssen folgende Schritte durchgeführt werden:

- Bilddaten aus den Grafikkarten der Bildberechnungs-Server auslesen
- Bilddaten komprimieren
- Daten versenden
- Bilddaten entpacken
- Pixel in die Grafikkarte des Darstellungs-Servers schreiben

Um eine optimale Systemauslastung zu erreichen, werden die zu übertragenden Bilder in Teilbilder zerlegt. Jedes Teilbild wird getrennt ausgelesen, komprimiert, versendet, entpackt und in den Bildspeicher des Zielrechners kopiert.

Die einzelnen Schritte beim Versenden der Teilbilder werden in Form einer parallelen Pipeline abgearbeitet. Während beispielsweise ein Teilbild für die Datenübertragung komprimiert wird, wird bereits das nächste Teilbild aus dem Speicher der Grafikkarte ausgelesen. Die Geschwindigkeit ist hierbei vom langsamsten Verarbeitungsschritt abhängig. Welcher Schritt dies ist, hängt sehr stark von der eingesetzten Hardware ab. Meist sind jedoch entweder die Übertragung über das Netzwerk oder das Schreiben des Bildspeichers auf dem Zielrechner die zeitaufwendigsten Arbeitsschritte. In Abbildung 5-7 ist ein Ablaufdiagramm zur Bildübertragung dargestellt. In diesem Beispiel ist die Performanz durch die Netzwerkübertragung limitiert.

5.6.2 Bildkompression

Die Optimierung des Bildkompositionsablaufs führt nur dann zu einem schnelleren Bildaufbau, wenn der langsamste Arbeitsschritt hierdurch beschleunigt wird. Die Datenübertragung über ein Netzwerk lässt sich beispielsweise durch eine Komprimierung der Daten beschleunigen. Hierbei muss jedoch berücksichtigt werden, dass der Arbeitsaufwand bei der Komprimierung nicht selbst zum Flaschenhals wird. Beim Einsatz von Gigabit-Netzwerken bringen komplexe Kompressionsverfahren keinen Geschwindigkeitsvorteil. Theoretisch lassen sich in solchen Netzen bis zu 30 Millionen Bildpunkte pro Sekunde übertragen. Nur wenige Kompressionsverfahren erreichen diese Geschwindigkeit bei der Kompression. Meist werden nur einfache Verfahren wie z.B. „Run-length encoding“ eingesetzt. Ahrens und Painter verwenden beispielsweise eine Run-Length-Kodierung zur Beschleunigung der Bildübertragung [AhP98]. Einfache und schnelle Verfahren haben jedoch den Nachteil, dass sie in komplexen Szenen nur sehr geringe Kompressionsraten erreichen. Insbesondere Szenen mit texturierten Objekten lassen sich mit Hilfe der Run-Length-Kodierung kaum komprimieren. Komplexere Kompressionsverfahren, wie sie beispielsweise bei der Kompression von JPEG-Bildern verwendet werden, erreichen zwar sehr gute Kompressionsraten, sind jedoch nicht für die Kompression von hochauflösenden Bildern für interaktive Bildwiederholraten geeignet.

Im hier vorgestellten Verfahren wird eine sehr einfache und schnelle Kompression verwendet. Ein Teilbild wird nur dann übertragen, wenn es Bildpunkte enthält, die nicht Teil des Bildhintergrundes sind. Damit lassen sich große Teile des Bildrandes herausfiltern, ohne die Übertragung der komplexen Bildinformationen durch eine ineffiziente Kompression auszubremsen.

5.6.3 Optimierung der Teilbildgröße

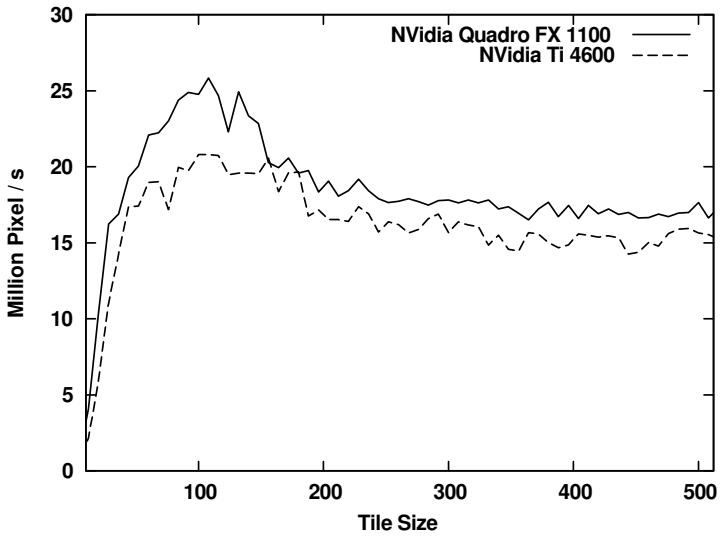

Abbildung 5-8: Übertragung von Bildpunkten bei unterschiedlichen Kachelgrößen

Eine weitere Möglichkeit zur Beschleunigung der Bildkomposition besteht darin, die Größe der Teilbilder so zu wählen, dass alle Stufen im Bildkompositionsablauf optimal bearbeitet werden. Werden beispielsweise die Teilbilder zu klein gewählt, dann lassen sich keine hohen Datenübertragungsraten im Netzwerk erreichen. Sehr große Teilbilder können dagegen dazu führen, dass die Verarbeitung aufgrund von begrenzten Pufferkapazitäten blockiert wird. Eine optimale Größe ist sehr stark von der eingesetzten Hardware abhängig. Abbildung 5-8 zeigt die Anzahl der übertragenen Bildpunkte bei unterschiedlichen Teilbildgrößen. Es werden zwei unterschiedlich schnelle Grafikkarten in einem Gigabit-Netzwerk untersucht. Im aufgeführten Test werden quadratische Teilbilder mit unterschiedlichen Kantenlängen übertragen. Bei einer Teilbildgröße von ca. 100x100 Bildpunkten wird das Maximum erreicht. Bei 26 Millionen Bildpunkten werden etwa 80 Megabyte an Daten übertragen. Diese Datenrate ließe sich mit einer nicht parallelisierten Bildübertragung nicht erreichen.

5.6.4 Vermeidung von Netzwerküberlastungen

Bei der dynamischen Lastverteilung für Tiled Displays mit vielen Einzelprojektionen müssen große Datenmengen zwischen den einzelnen Rechnern effizient ausgetauscht werden. Wird zur Übertragung das TCP/IP-Protokoll verwendet, können durch eine Überlastung der Netzwerkverbindung Verzögerungszeiten entstehen. Erkennt das Protokoll die Überlastung einer

Netzwerkverbindung, so werden von den Sendern Übertragungspausen erzwungen. Diese Pausen können so lang sein, dass sie eine flüssige Interaktion unmöglich machen.

In dem hier vorgestellten Verfahren können dann Überlastungen auftreten, wenn ein Server von mehreren anderen Servern Bildteile zugesendet bekommt. Da allen Rechnern in einem Cluster meist die gleiche Bandbreite zur Verfügung steht, ist es sehr wahrscheinlich, dass bereits die Bildteile zweier parallel sendender Server die Netzwerkbandbreite eines empfangenden Servers übersteigen.

Um dieses Problem zu lösen, wird die Kommunikation für die Bildzusammenführung den speziell gegebenen Anforderungen entsprechend umgesetzt. Jeder Server erhält das Wissen darüber, von welchen anderen Servern er Bilddaten empfangen muss. Diese fordert er sequentiell nacheinander an. Dieses Vorgehen kann eine Überlastung der Netzwerkverbindung zuverlässig verhindern. Eine weitere Optimierung besteht darin, dass eine Anforderung nicht erst dann gesendet wird, wenn die vorhergehende Übertragung beendet ist, sondern bereits eine kurze Zeitspanne davor. Damit kann der Empfangskanal auch während der Laufzeit der Folgeanfrage ausgelastet werden.

5.7 Anbindung von Bildkompositions-Hardware

Mit Sepia [MHS99], Metabuffer [BBF00] oder Lightning-2 [SEP01], Orad DVG oder dem Netzwerk-Bildspeicher von IBM wurden in den letzen Jahren einige Systeme entwickelt, die in einem Cluster aus Standardkomponenten eine beschleunigte Bildkomposition durchführen. Bei allen Lösungen wird die Bildsynthese durch eine spezielle Hardware durchgeführt. Die erreichbaren Bildwiederholraten liegen bei einem Hardware-Ansatz sehr nahe an der Bildwiederholrate von etwa 60 Hz. Es ist abzusehen, dass in Zukunft weitere leistungsfähige und preiswerte Lösungen am Markt verfügbar sein werden. Aus diesem Grund werden die oben beschriebenen Verfahren so in ein Rahmensystem integriert, dass die Bildkomposition entweder per Software über das lokale Netzwerk oder mit spezieller Hardware durchgeführt werden kann. Das Rahmensystem lässt sich auf einfache Weise um weitere Treiber für neue Hardware erweitern. Die dynamische Lastverteilung kann für alle Kompositionsvarianten unverändert verwendet werden.

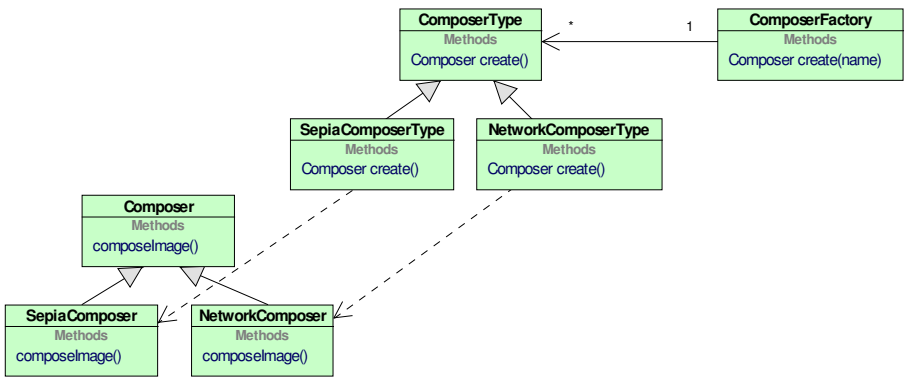

Abbildung 5-9: Framework für die Bildkomposition

Die Erweiterbarkeit des Rahmensystems wird mit Hilfe des „Factory Method“-Entwurfsmusters realisiert. Alle Verfahren zur Bildkomposition werden von einer gemeinsamen Basisklasse abgeleitet. Dies gilt sowohl für die oben beschriebene Bildkomposition über das Netzwerk als auch für die Bildkomposition mit Hardwareunterstützung. Jedes Verfahren zur Bildkomposition registriert sich in einer zentralen Funktionstabelle. Die Funktionstabelle kann zur Laufzeit erweitert werden, ohne dass zentrale Komponenten des Rahmensystems angepasst werden müssen. Jedes Kompositionsverfahren registriert sich mit einem symbolischen Namen. Über diesen Namen kann ein Anwender ein Verfahren auswählen. Wird vom Anwender keine Vorauswahl getroffen, dann wird die Funktionstabelle automatisch nach einem geeigneten Verfahren durchsucht. Jedes Bildkompositionsobjekt stellt für diese Suche eine Test-Methode zur Verfügung, in der geprüft wird, ob die erforderliche Hardware vorhanden ist. Schlägt der Test fehl, dann wird das nächste Verfahren in der Funktionstabelle untersucht.

5.8 Ergebnisse

Der in dieser Arbeit neu entwickelte Algorithmus zur dynamischen Lastverteilung wird im Folgenden anhand einiger Testszenen auf seine Verwendbarkeit getestet. Das Haupteinsatzgebiet liegt in der dynamischen Lastverteilung auf zusammengesetzten Projektionssystemen. Es ist bekannt, dass für die Parallelisierung der Bildberechnung für ein einzelnes Display effizientere Ansätze existieren, die nicht mit den Problemen einer Sort-First-basierten Parallelisierung behaftet sind und daher besser verteilen. Darauf wird in dem folgenden Kapitel genauer eingegangen. Dennoch bietet das hier entwickelte Verfahren einen guten Kompromiss in Bezug auf Skalierung und Kommunikationsaufwand.

Es wurden drei Testszenen mit unterschiedlichen Eigenschaften ausgewählt. Alle drei Szenen lassen sich ohne eine Parallelisierung auf aktueller Hardware nicht in interaktiven Bildwiederholraten darstellen. Die einzelnen Modelle unterscheiden sich jedoch sehr stark in ihren Eigenschaften und Anforderungen an die Bildberechnungshardware.

Abbildung 5-10: Stanford Dragon mit 8 Millionen Polygonen

Abbildung 5-11: UNC Power Plant mit 13 Millionen Polygonen

Abbildung 5-12: BMW 7 mit 4 Millionen Polygonen und Shader-Programmen

- **Standford Dragon**: Das Modell eines Drachen stammt aus dem „Stanford Scanning Repository“. Es wurde mit einem 3D-Scanner erfasst und besitzt etwa acht Millionen Polygone. Die Polygone sind sehr gleichmäßig über das gesamte Modell verteilt. Um das Modell für ein Szenengraphensystem handhabbar zu machen, wurde das Modell in 1024 Einzelobjekt zerlegt. Die Berechnungszeit ist durch die Transformationsleistung der Grafikhardware bestimmt.

- **Power Plant**: Dieses Modell stammt von der University of North Carolina und war lange Zeit eines der größten frei verfügbaren Modelle. Es besteht aus 13 Millionen Polygonen. Die Polygone sind jedoch sehr ungleichmäßig über das Modell verteilt. Ein sehr großer Teil der Komplexität ist in einem kleinen Teil des Modells konzentriert. Dies ist auch deutlich in Abbildung 5-2 zu erkennen.

- **BMW 6**: Dieses Modell besitzt etwa vier Millionen Polygone. Diese sind relativ gleichmäßig verteilt. Das Modell besteht aus 800 Einzelobjekten, deren Größe jedoch sehr stark variiert. Beispielsweise sind große Teile der Karosserie in einem einzelnen Objekt zusammengefasst. Die Berechnungszeit wird hauptsächlich durch die komplexe Materialeigenschaft der Karosserie bestimmt.

Abbildung 5-10 bis Abbildung 5-12 zeigen die Modelle und einen Animationspfad, der zur Ermittlung der Laufzeiten verwendet wurde. Auch wenn die Parallelisierung für ein einzelnes Display nicht das Hauptziel der Entwicklung darstellt, werden im Folgenden zuerst die Ergebnisse für die parallele Berechnung für ein einzelnes Display aufgeführt.

5.8.1 Lastverteilung für ein einzelnes Display

Mit Hilfe der hier entwickelten dynamischen Lastverteilung lassen sich bereits mit vier PCs interaktive Bildwiederholraten erreichen. Die Skalierung ist nicht linear und wird auch mit steigendem Parallelisierungsgrad schlechter. Dies war jedoch auch zu erwarten und ist ein Problem aller Sort-First-basierten Verfahren. Es ist allerdings auch zu erkennen, dass der Mehraufwand, der durch das neue Verfahren verursacht wird, sehr gering ist. Dies ist daran zu erkennen, dass mit zwei PCs fast eine Verdopplung der Performanz erreicht wird. Auf diesen Aspekt wird später noch genauer eingegangen. Das „Power Plant“-Modell lässt sich mit Hilfe von Sort-First nur sehr schwer beschleunigen. Bei der Aufteilung des Bildbereiches müssen sehr kleine und dennoch komplexe Bildbereiche geteilt werden. Die Ergebnisse zeigen, dass dies mit dem entwickelten Verfahren gelingt. Die erzielten Bildwiederholraten müssen auch im Verhältnis zu der maximal erreichbaren

Performanz betrachtet werden. In Abbildung 5-2 wurde gezeigt, dass dieses Modell mit Hilfe eines Sort-First-basierten Ansatzes nicht schneller als 29 mal pro Sekunde gezeichnet werden kann. Um dies Ergebnis zu erreichen, sind mehrere 100 parallel rechnende PCs erforderlich. Unter diesen Voraussetzungen sind die 6,5 Bilder pro Sekunde mit 16 PCs als gutes Ergebnis zu werten. Auch für das BMW-Modell liefert das Verfahren gute Ergebnisse. Die Bildwiederholrate lässt sich bereits mit 16 PCs um den Faktor vier beschleunigen.

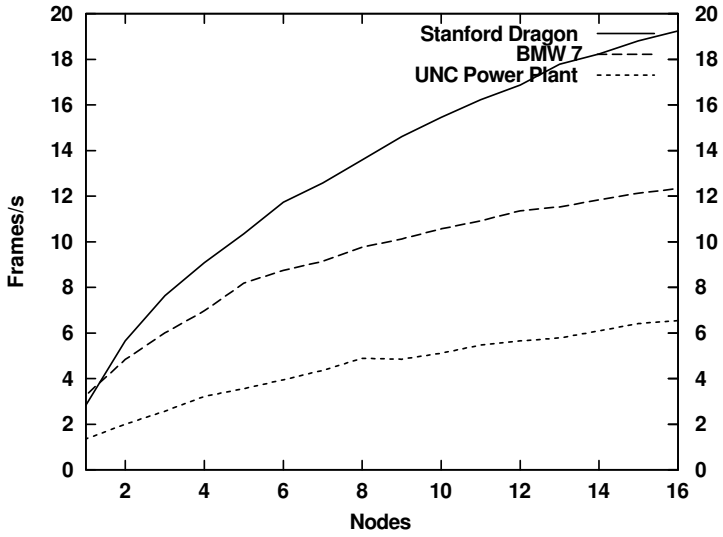

Abbildung 5-13: Beschleunigung Stanford Dragon

Zusammenfassend kann festgestellt werden, dass das neu entwickelte Verfahren in der Lage ist, die Bildberechnung wesentlich zu beschleunigen und auch komplexe Szenen für interaktive Anwendungen nutzbar zu machen.

5.8.2 Einfluss der Kostenfunktion

In diesem Abschnitt soll untersucht werden, welchen Einfluss die Kostenfunktion auf das Ergebnis der Lastverteilung hat. Meist werden bei der Lastverteilung nur die sichtbaren Dreiecke ermittelt. In dem hier vorgestellten Ansatz werden auch die Kosten für die Transformation und die Kosten für das setzen von Bildpunkten berücksichtigt. Es soll untersucht werden, wie sich diese erweiterte Kostenermittlung auf die Performanz auswirkt. Dazu werden zwei Modelle jeweils viermal auf 1 bis 32 Rechnern parallel berechnet. Die dargestellten Diagramme zeigen die erreichte Performanz mit unterschiedlichen Lastverteilungsfunktionen. Es wird einmal die vollständige Lastfunktion und anschließend in drei getrennten Testläufen jeweils eine Komponente der Lastfunktion verwendet.

Abbildung 5-14 zeigt die Performanz bei der parallelen Berechnung des „Stanford Dragon“-Modells. Beim Einsatz der vollständigen Kostenfunktion werden die höchsten Bildwiederholraten erreicht. Es ist jedoch auch zu erkennen, dass die Darstellung des Modells im wesentlichen durch die fixen Kosten bestimmt wird.

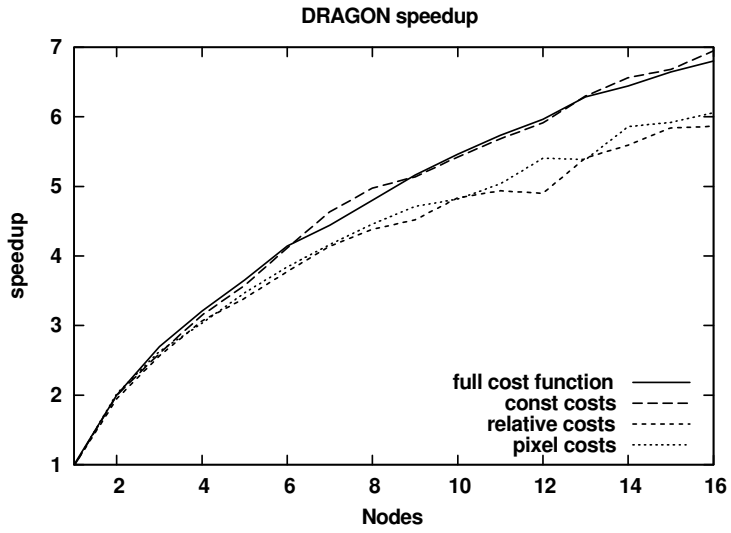

Abbildung 5-14: Einfluss unterschiedlicher Lastfaktoren (Dragon)

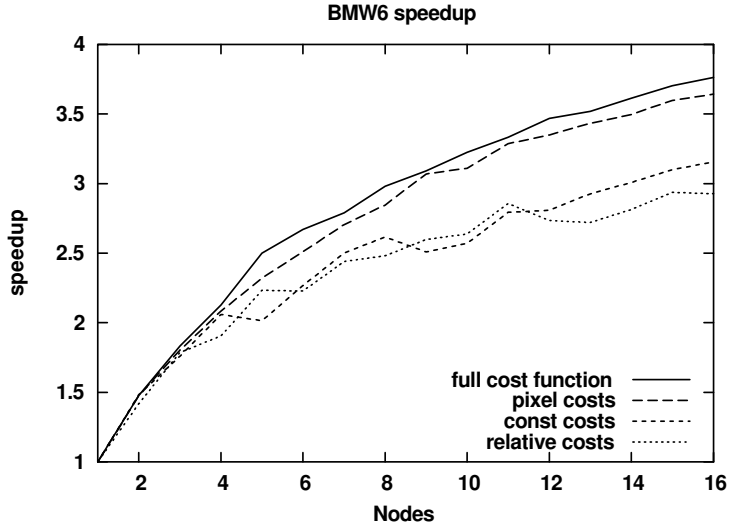

Abbildung 5-15: Einfluss unterschiedlicher Lastfaktoren (BMW6)

Das BMW-Modell wird aufgrund seiner Materialeigenschaften sehr stark durch die Kosten für das Setzen der Bildpunkte bestimmt. Dies ist auch deutlich aus Abbildung 5-15 abzulesen. Auch hier ist deutlich zu erkennen, dass die vollständige Kostenfunktion das beste Resultat liefert.

Aus den gezeigten Ergebnissen ist zu erkennen, dass die verwendete Lastfunktion in der Lage ist, gute Ergebnisse für sehr unterschiedliche Modelle zu liefern. Es hat sich auch gezeigt, dass durch die Einbeziehung der konstanten Kosten Szenen mit komplexer Geometrie, schneller dargestellt werden können.

5.8.3 Lastverteilung für ein Tiled Display

In diesem Abschnitt werden die oben beschriebenen Testszenen auf einem Tiled Display mit 24 Projektoren dargestellt. Es wird untersucht, wie sich die Bildwiederholrate bei einer Darstellung

ohne Lastverteilung, mit Lastverteilung und unter der zusätzlichen Einbeziehung von 24 weiteren PCs verhält. Bei der Darstellung eines Bildes auf einem Tiled Display handelt es sich bereits um eine Sort-First-Parallelisierung. Aus diesem Grund sind die Bildwiederholraten auch ohne eine dynamische Lastverteilung bereits höher als die zuvor ermittelten Werte für ein einzelnes Display.

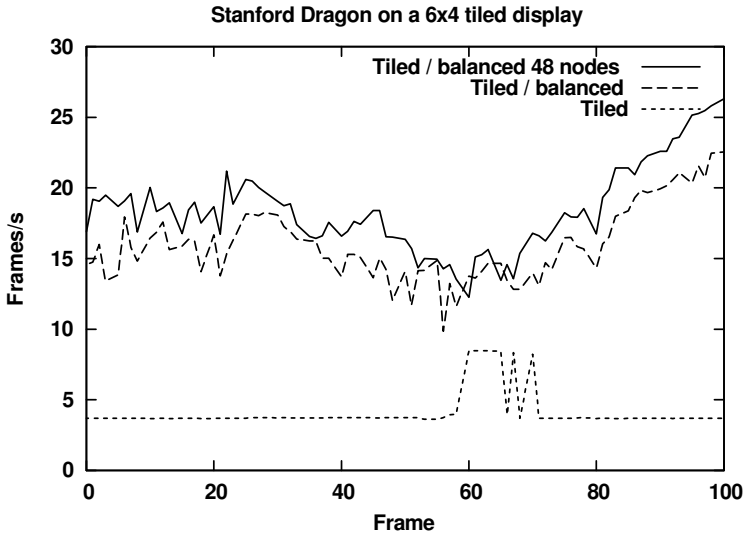

Abbildung 5-16: Stanford Dragon auf einem 6x4-Tiled Display

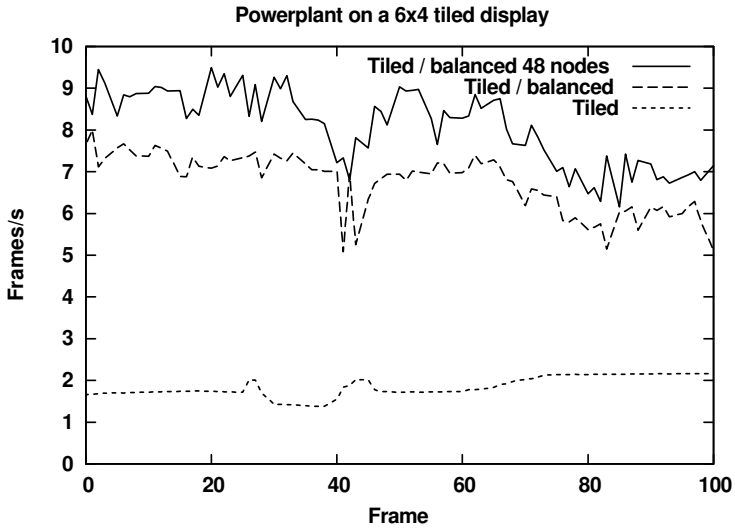

Abbildung 5-17: Power Plant auf einem 6x4-Tiled Display

Das „Stanford Dragon“-Modell lässt sich mit Hilfe der neu entwickelten Lastverteilung mit mehr als zehn Bildern pro Sekunde darstellen. Obwohl bei der Darstellung keine zusätzliche Hardware eingesetzt wird, kann die Darstellungsgeschwindigkeit allein durch die bessere Auslastung der Ressourcen um den Faktor vier und mehr beschleunigt werden. Die Performanz kann durch den Einsatz weiterer PCs noch gesteigert werden, doch fällt diese Steigerung sehr gering aus.

Auch das Power-Plant-Modell kann durch den Einsatz der dynamischen Lastverteilung zwei- bis dreimal so schnell dargestellt werden.

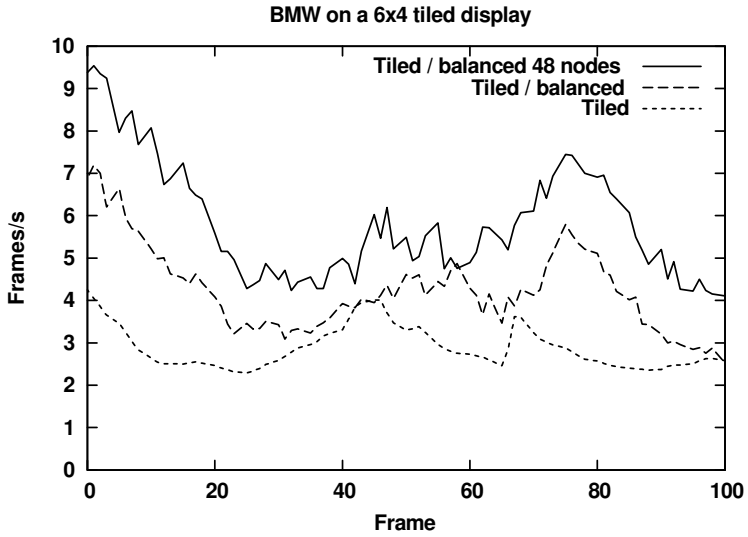

Abbildung 5-18: BMW6 auf einem 6x4-Tiled Display

Das BMW-Modell wird ebenfalls spürbar beschleunigt. Der Beschleunigungsfaktor fällt jedoch relativ gering aus. Dies liegt im wesentlichen daran, dass während der gesamten Animation das Auto auf vielen Kacheln gleichzeitig zu sehen ist. Da die komplexen Materialeigenschaften der Geometrie die Berechnungszeit limitieren, bleibt nur ein geringer Spielraum für eine Performanz-Steigerung. Durch den Einsatz zusätzlicher PCs kann die Bildwiederholrate weiter gesteigert werden.

5.8.4 Aufschlüsselung der Berechnungszeiten

In diesem Abschnitt soll untersucht werden, wie sich die Berechnungszeiten beim Einsatz der dynamischen Lastverteilung zusammensetzen.

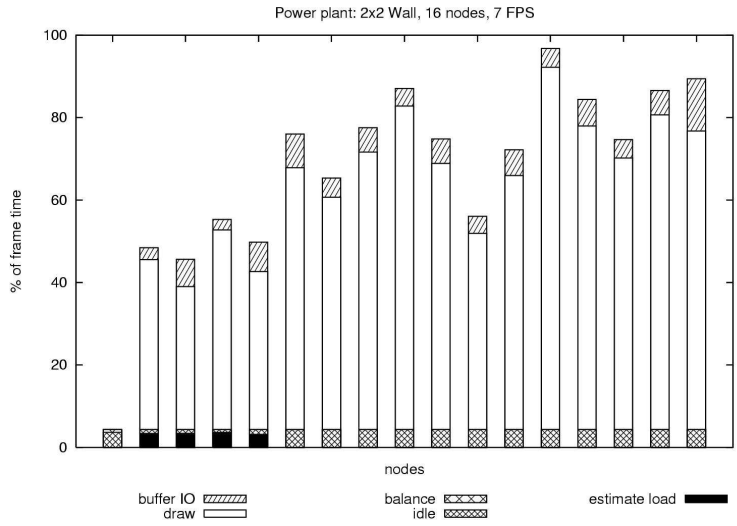

Abbildung 5-19: Darstellung der Power Plant auf 2x2-Displays mit 32PCs

Abbildung 5-19 zeigt, dass die Zeit zur Berechnung eines Bildes im wesentlichen durch das Zeichnen der Szene in den Bildspeicher bestimmt ist. Die Zeiten für die Lastermittlung und

Verteilung sind im Gegensatz dazu sehr gering. Der Bildberechnungsaufwand wird auf alle Rechner verteilt. Die ungleiche Auslastung ergibt sich aus der ungenauen, aber dafür schnellen Lastabschätzung und der Struktur des Power-Plant-Modells, dass sich nur schwer in Bereiche gleicher Last einteilen lässt.

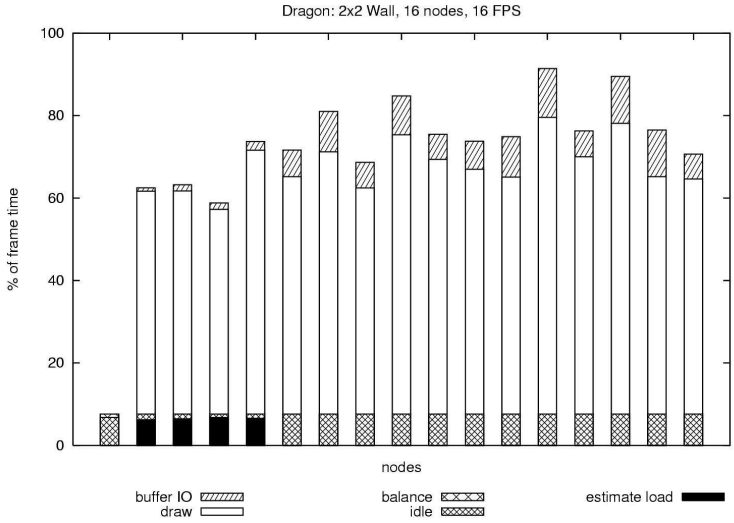

Abbildung 5-20: Dragon-Modell auf einem 2x2-Display mit 32 PCs

Das Dragon-Modell lässt sich einfacher in Bereiche gleicher Last einteilen. Dies ist an der gleichmäßigeren Auslastung der Rechner zu erkennen. Bei diesem Modell spielen die Zeiten zur Lastermittlung und Verteilung so gut wie keine Rolle. Die Berechnungszeit wird durch die Bildberechnung und die Netzwerkkommunikation bestimmt.

5.9 Zusammenfassung

Es wurde eine neuer Algorithmus zur dynamischen Lastverteilung auf der Basis von Sort-First entwickelt. Das Verfahren erreicht trotzt der beschränkten Skalierbarkeit von Sort-First gute Ergebnisse. Im Vergleich zu anderen Ansätzen wurde sowohl die Lastermittlung als auch die Lastverteilung verbessert.

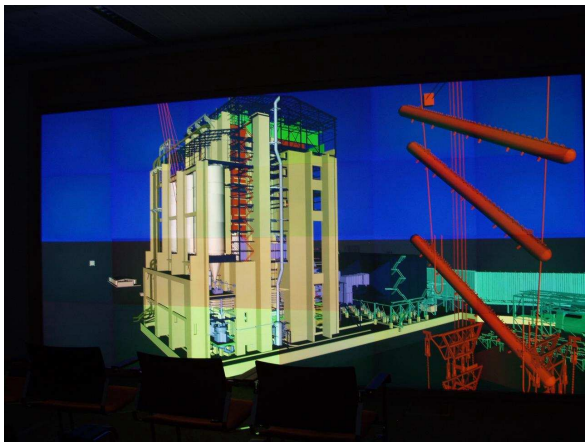

Abbildung 5-21: Power-Plant-Modell auf der HEyeWall

Im Gegensatz zu bestehenden Verfahren wird bei der Lastermittlung auch der Teil der Bildberechnung mit einbezogen, der nicht unmittelbar von der Sichtbarkeit eines Objektes abhängig ist. Mit diesen zusätzlichen Informationen ist der Algorithmus zur Lastverteilung in der Lage, den Bildbereich so zu unterteilen, dass möglichst wenige Objekte des Szenengraphen durch Bereichsgrenzen geschnitten werden.

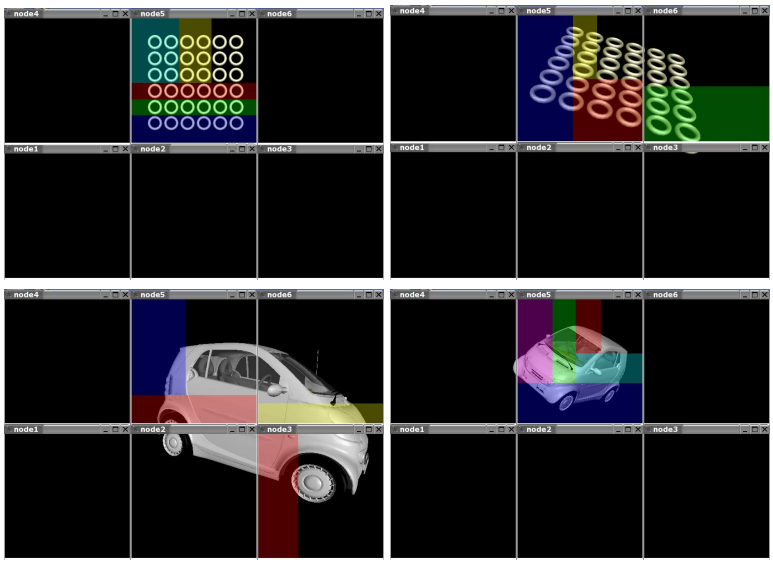

Abbildung 5-22: Ergebnisse der dynamischen Lastverteilung

Die Lastverteilung ist in der Lage, die Berechnungszeit für ein einzelnes Display auf viele PCs zu verteilen. Es ist jedoch auch möglich, die Last in einem zusammengesetzten Projektionssystem zwischen den Einzelprojektionen auszugleichen. Für jede Konfiguration können zusätzliche PCs einbezogen werden, um eine weitere Performanz-Steigerung zu erreichen.

In einem späteren Kapitel wird genauer darauf eingegangen, wie die hier entwickelten Techniken ohne Änderung auch für die Lastverteilung in einer Cave oder für ein beliebiges anderes Projektionssystem mit mehreren Einzelprojektionen eingesetzt werden können.

6 Sort-Last-parallele Bildberechnung

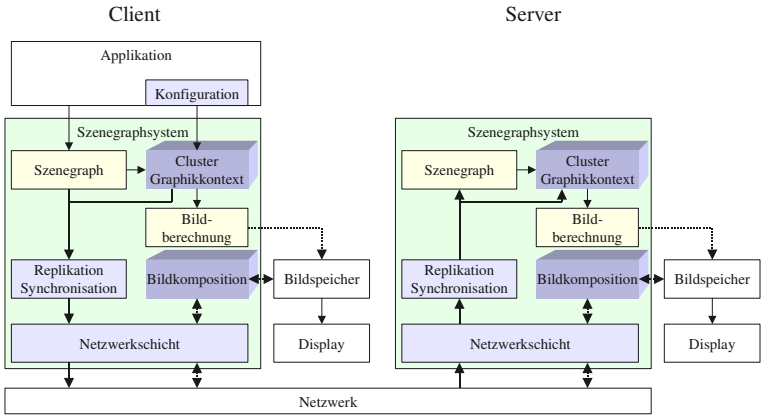

Abbildung 6-1: Sort-Last-parallele Bildberechnung

Bei parallelen Bildberechnungsverfahren auf der Basis von Sort-Last wird die Szene aufgeteilt, und anschließend werden alle Teile parallel berechnet. Im Gegensatz zu Sort-First lassen sich hierbei sehr gute Skalierungsfaktoren erreichen. Da bei der Berechnung der Teilszenen keine Informationen zwischen den parallel arbeitenden Instanzen ausgetauscht werden müssen und auch keine Teile mehrfach verarbeitet werden, ist es möglich, auf der Basis von Sort-Last eine ideale Skalierbarkeit zu erreichen. Diesem Vorteil stehen jedoch einige Nachteile gegenüber:

Da die Teilszenen unabhängig voneinander berechnet werden, ist es schwierig, transparente Flächen korrekt darzustellen. Szenengraphensysteme wie OpenSG oder Performer sortieren transparente Flächen in der Bildebene von hinten nach vorne. Werden nun Teile der Szene unabhängig voneinander berechnet, ist eine globale Sortierung nicht mehr möglich.

Eine Szene wird in mehrere Teilszenen unterteilt und parallel berechnet. Hierbei entstehen mehrere Einzelbilder, die jeweils nur einen Teil der Szene darstellen. Um eine korrekte Darstellung der gesamten Szene zu erhalten, müssen alle Teilergebnisse miteinander kombiniert werden. Für jeden Bildpunkt muss der Farbwert übernommen werden, der dem Betrachter am nächsten liegt. Das bedeutet, dass für die Bildkomposition nicht nur die Farbwerte aller Einzelbilder, sondern auch die Tiefeninformation aller Bildpunkte in allen Teilbildern berücksichtigt werden müssen. Hierbei müssen sehr große Datenmengen zusammengeführt werden. Die Bildkomposition ist das Hauptproblem beim Einsatz eines Sort-Last-basierten Verfahrens. Selbst mit modernsten Netzwerktechniken lässt sich die anfallende Datenmenge nicht ohne Probleme bewältigen. Aus diesem Grund wird im Folgenden ein ausführlicher Vergleich bestehender Bildkompositionsverfahren durchgeführt. Aus den hierbei gewonnenen Erkenntnissen wird ein neues, speziell für Cluster-Systeme optimiertes Bildkompositionsverfahren entwickelt, mit dessen Hilfe interaktive Bildwiederholraten auch mit handelsüblichen Netzwerktechniken erreicht werden.

Der folgende Abschnitt geht kurz auf die Lastverteilung bei Sort-Last ein. Im Anschluss daran folgt eine ausführliche Beschreibung der Bildkomposition.

6.1 Lastverteilung bei Sort-Last

Das Problem der Lastverteilung bei Sort-Last besteht darin, eine Szene so zu unterteilen, dass alle parallel arbeitenden Rechner den gleichen Arbeitsaufwand zu bewältigen haben. Die Lastverteilung kann dynamisch für jedes Bild erfolgen oder statisch beim Laden einer neuen Szene. Die dynamische Lastverteilung kann genauere Ergebnisse liefern, da hierbei die Blickrichtung der virtuellen Kamera bei der Lastverteilung berücksichtigt werden kann. Objekte, die im aktuellen Bild nicht sichtbar sind, müssen bei der Aufteilung nicht berücksichtigt werden. Die dynamische Lastverteilung hat jedoch den Nachteil, dass die einzelnen Rechner von Bild zu Bild andere Teile der Szene darstellen müssen. Dies führt dazu, dass eventuell Texturen ausgelagert oder grafische Objekte neu in den Speicher der Grafikhardware geladen werden müssen. Zusätzlich muss jeder Rechner einen Zugriff auf die vollständige Szene haben. In [SFL01] wird eine optimierte Verteilung für eine dynamische Lastverteilung vorgeschlagen. Teile der Szene werden nicht mehr auf alle, sondern nur auf einige Rechner übertragen. Die Lastverteilung berücksichtigt diese Verteilung. Da Sort-Last häufig für sehr große Szenen mit vielen Objekten eingesetzt wird, ist für die dynamische Lastverteilung meist ein erheblicher Rechenaufwand erforderlich. Aufgrund dieser Nachteile wird in der hier vorliegenden Arbeit eine statische Aufteilung verwendet.

Wie oben beschrieben, besteht das Hauptproblem bei einem Sort-Last-Verfahren in der Bildkomposition der einzelnen Teilbilder. Dieser Aufwand kann erheblich reduziert werden, wenn bei der Bildkomposition die Teile des Bildes ignoriert werden, in denen keine Objekte zu sehen sind. Diese Bereiche lassen sich in einem Szenengraph sehr schnell mit Hilfe des Hüllvolumens bestimmen. Je kleiner das Hüllvolumen einer Teilszene ist, desto kleiner erscheint das Objekt in der Bildebene. Aus diesem Grund wird im hier vorgestellten Verfahren eine statische Unterteilung der Szene vorgenommen, bei der jede Teilszene ein möglichst kleines Hüllvolumen besitzt.

6.2 Analyse bestehender Kompositionsverfahren

Bei allen parallelen Bildberechnungsverfahren, die auf dem Sort-Last-Prinzip basieren, werden mehrere Bilder berechnet, die jeweils nur einen Teil der Szene darstellen. Aus diesen einzelnen Bildern muss das Ergebnisbild berechnet werden, so dass alle Teile der Szene in einem einzigen Bild enthalten sind. Die Einzelbilder können Objekte in beliebigen Entfernungen zum Betrachter darstellen. Aus diesem Grund sind für die korrekte Zusammenfügung des Ergebnisbildes Informationen über die Tiefe der Pixel erforderlich. In das Ergebnisbild wird für jeden Bildpunkt genau das Pixel aus den einzelnen Bildern übernommen, welches dem Betrachter am nächsten liegt.

Die Entfernung kann auf Pixel-Ebene aus dem Z-Buffer entnommen werden, der beim Zeichnen der Teilszenen gefüllt wird. Wird die Szene in räumlich nicht überlappende Bereiche aufgeteilt, kann bei der Bildkomposition auf eine Entfernung auf Pixel-Ebene verzichtet werden. Das

Ergebnisbild wird in diesem Fall durch Überlagerung der Einzelbilder von hinten nach vorne erzeugt [NgZ00].

Für die Bildkomposition müssen sehr große Datenmengen bewältigt werden. Am Beispiel von 32 Rechnern, die gemeinsam ein Bild mit einer Auflösung von 1024 x 768 Bildpunkten berechnen, soll dies verdeutlicht werden. Bei einer Standardfarbtiefe von 24 Bit und einer Z-Buffer-Genauigkeit von 32 Bit werden für jeden Bildpunkt 7 Bytes benötigt. Für ein komplettes Bild müssen etwa 5,5 MByte übertragen werden. Soll nun aus den 32 einzelnen Bildern ein Ergebnisbild berechnet werden, so müssen die Daten von mindestens 31 Bildern zwischen den einzelnen Rechnern ausgetauscht werden. Für ein Ergebnisbild müssen also etwa 170 MByte übertragen werden. Sollen nun Bilder in interaktiven Bildwiederholraten berechnet werden, muss die Kommunikationsinfrastruktur ein Datenvolumen von etwa 32 GBit/s bewältigen.

Eine Möglichkeit, die Datenmenge zu reduzieren, besteht darin, nur die Bildpunkte zur Komposition zu versenden, die von einem Objekt verdeckt werden. Meist wird für die Bestimmung aktiver Punkte ein umschließendes Rechteck um einzelne Polygone oder um Gruppen von Polygonen verwendet. Wenn nicht alle Bildpunkte für die Bildkomposition verwendet werden, spricht man auch von einer Sort-Last-Sparse-Architektur im Gegensatz zu einer Sort-Last-Full-Architektur [MCE94,YCC99]. Der Erfolg dieser Strategie hängt stark von der Lokalität der Objekte ab, die von den einzelnen Rechnern gezeichnet werden.

Das Datenvolumen lässt sich auch durch Kompression reduzieren. Häufig werden hierfür einfache und schnelle Algorithmen eingesetzt. Beispielsweise wird in [LaL94] und [AhP98] eine Run-Length-Kodierung verwendet, um die Bilder verlustfrei zu komprimieren. Es muss jedoch ein Kompromiss zwischen Kompressionsrate und Kompressionsaufwand gefunden werden. Eine Kompression ist nur dann sinnvoll, wenn sie den erforderlichen Rechenaufwand durch Einsparungen an Datenübertragungszeit aufwiegt. Der sinnvolle Einsatz von Kompressionsverfahren ist abhängig vom Verhältnis der CPU-Geschwindigkeit zur Netzwerkgeschwindigkeit. Mögliche Kompressionsverfahren sind z.B. das verlustbehaftete JPEG-Format oder ein verlustfreies Verfahren ähnlich dem, wie es beispielsweise in [MAM95, MAM97] beschreiben wird.

Die sehr großen Datenmengen, die zur Bildkomposition versendet werden müssen, können auch durch eine Kompression nicht in dem Maße verringert werden, dass die Datenübertragung mit geringem Hardware-Aufwand zu bewältigen ist. Um dies zu erreichen, wurden verschiedene Verfahren entwickelt, die den Arbeitsaufwand auf möglichst viele parallel arbeitende Prozessoren und Kommunikationswege verteilen. Ein Großteil der Forschung auf diesem Gebiet erfolgte im Kontext der Darstellung von Volumen-Datensätzen [Neu93, MPH94, LaL94, Law96, YYC99, ZBB01, LMS01]. Da diese Ergebnisse zum größten Teil auf die Darstellung von Polygonen übertragen werden können, wurden bisher nur wenige spezielle Untersuchungen für die Bildkomposition bei polygonalen Modellen vorgenommen [LRN94,KAO98].

Die folgenden Abschnitte beschreiben die wichtigsten bisher entwickelten Verfahren zur Bildkomposition. Um den Grad der Parallelität zu veranschaulichen, wird für jedes Verfahren neben dem aufsummierten Datenvolumen auch das maximale sequentiell zu übertragende Datenvolumen angegeben. Ist die Kommunikationsinfrastruktur parallelisiert, so errechnet sich der Zeitaufwand für die Datenübertragung aus der Übertragungsrate multipliziert mit der sequentiell zu übertragenden Datenmenge.

Neben dem hier vorgestellten Vergleich wird auch in den Arbeiten von James Painter u.a. [MPH94], Tong-Yee Lee u.a. [LRN96] oder Don-Lin Yang [YCC99] ein Überblick unterschiedlicher Verfahren zur Bildkomposition präsentiert. Allerdings sind diese Vergleiche nicht vollständig oder teilweise beschränkt auf spezielle Hardwarearchitekturen.

Da es sich bei den untersuchten Verfahren meist nicht um spezialisierte Verfahren für Cluster-Systeme handelt, wird im Folgenden von parallelen Prozessoren und nicht von parallel arbeitenden Rechnern gesprochen. Die gewonnenen Erkenntnisse lassen sich sowohl auf Multiprozessorsysteme als auch auf Cluster-Systeme übertragen. Für den hier vorgestellten Vergleich werden die folgenden Bezeichnungen und Symbole verwendet:

n	Anzahl der Prozessoren
A	Anzahl der Bildpunkte
A_{tile}	Größe der Teilbilder bei der Pipeline-Komposition
d_{seq}	Nicht parallel übertragene Datenmenge
m_{seq}	Anzahl der sequentiell übertragenen Nachrichten
d_{sum}	Summe aller übertragenen Daten

Table 1: Verwendete Symbole und Abkürzungen

6.2.1 Zentrale Bildkomposition

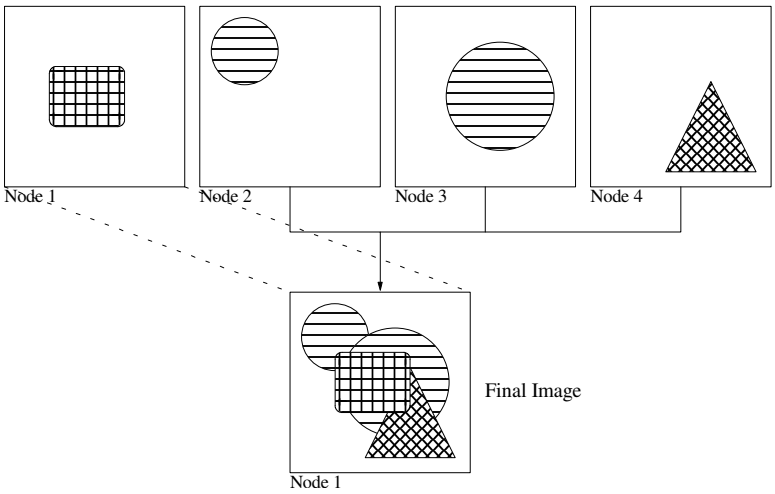

Abbildung 6-2: Zentrale Image-Komposition

Bei der zentralen Bildkomposition empfängt ein Prozessor alle Einzelbilder der anderen Prozessoren. Die Bildkomposition wird von einem einzigen Prozessor sequentiell durchgeführt. Die sequentiell zu lesende Datenmenge steigt linear mit der Anzahl der Prozessoren. Die Beschreibung einer Implementierung ist in der Arbeit von Nguyen und Zahorjan zu finden. Aufgrund der schlechten Skalierbarkeit wird dieses Verfahren nur selten eingesetzt.

Bei diesem Verfahren sind das summierte und das sequentiell zu übertragende Datenvolumen identisch. Es ist eine Anbindung aller Prozessoren an einen zentralen Prozessor zur Bildkomposition erforderlich. Neben einer Sterntopologie kann auch ein Bussystem verwendet werden, da keine parallele Kommunikation stattfindet.

$$d_{seq} = A(n-1)$$
$$m_{seq} = n-1$$
$$d_{sum} = d_{seq}$$

Die zentrale Bildkomposition hat den Vorteil, dass das fertige Bild komplett bei einem Prozessor zur Verfügung steht. Dies ist bei einigen der folgenden Verfahren nicht der Fall.

6.2.2 Binäre Bildkomposition

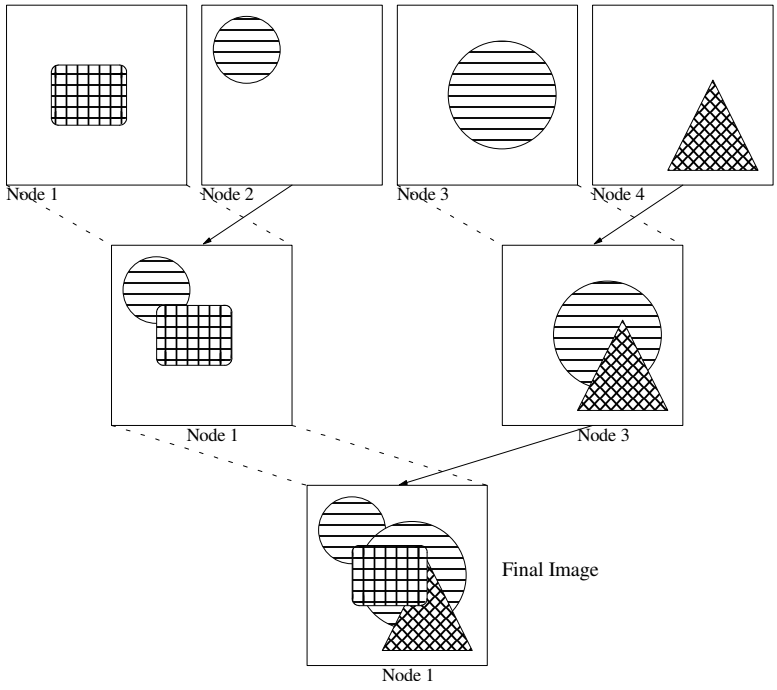

Abbildung 6-3: Binäre Bildkomposition

Bei der binären Bildkomposition handelt es sich um ein iteratives Verfahren. Es werden Paare von Prozessoren gebildet. Ein Prozessor sendet dem anderen sein komplettes Bild. Der Prozessor, der

das Bild des anderen empfängt, führt mit seinem eigenen Bild eine Bildkomposition durch. Alle Prozessorpaare führen diese Verarbeitung parallel durch. Die Anzahl der Prozessoren, deren Bilder zusammengefügt werden müssen, ist nun nur noch halb so groß wie vorher. Auf die verbleibenden Prozessoren wird der Iterationsschritt erneut angewandt. Dies wird so lange wiederholt, bis das fertige Bild an einem Prozessor vorliegt.

$$d_{seq} = A \log_2 n$$
$$m_{seq} = \log_2 n$$
$$d_{sum} = A(n-1)$$

Die Summe der übertragenen Pixeldaten ist bei diesem Verfahren so groß wie bei der zentralen Bildkomposition. Allerdings werden nun mehrere Datenströme parallel übertragen.

Dieses Verfahren wurde u.a. von R. Heiland [Hei94] und Ramakrishnan u.a. [RaS99] umgesetzt. Wie in [MPH94, LRN96] beschrieben, beruht die Ineffizienz dieses Verfahrens darauf, dass mit jedem Iterationsschritt mehr Prozessoren untätig werden. Es bleiben also Ressourcen ungenutzt. Es wurden daher weitere Verfahren entwickelt, um einen höheren Parallelisierungsgrad zu erreichen.

6.2.3 Direct-Send

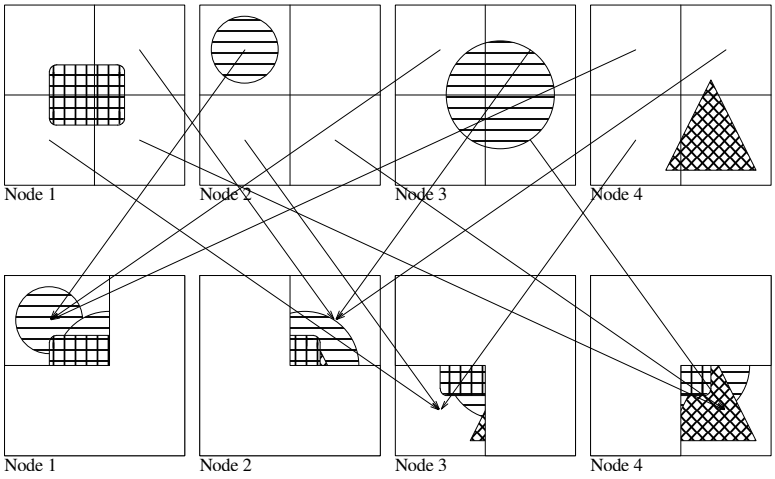

Abbildung 6-4: Direct-Send

Der Begriff „Direct-Send“ wurde von Kwan-Liu Ma u.a. in [MPH94] eingeführt. Bei dieser Form der Bildkomposition wird das Bild in Bereiche unterteilt. Jedem Prozessor wird die Verantwortung für einen Bereich zugewiesen. Alle Prozessoren schicken die Pixel der Bereiche, für die sie selbst nicht verantwortlich sind, an den jeweils zuständigen Prozessor, d.h., jeder Prozessor erhält von allen anderen Prozessoren die Pixel für den Bereich, für den er zuständig ist. Wenn der komplette Bildbereich einem einzigen Prozessor zugewiesen wird, ist dieses Verfahren identisch mit der zentralen Bildkomposition. Meist wird das Bild jedoch in gleich große Bereiche unterteilt, um eine maximale Parallelität zu erreichen.

Dieses Verfahren wurde u.a. von Neumann [Neu93] und Moreland u.a. [MWP01] verwendet. Die folgende Abschätzung der Datenmengen geht davon aus, dass jedem Prozessor ein gleich großer Bereich zugewiesen wird.

$$d_{seq} = A\left(1 - \frac{1}{n}\right)$$
$$m_{seq} = n - 1$$
$$d_{sum} = n d_{seq} = A(n-1)$$

Die von Ulrich Neumann in [Neu93] verwendete Abschätzung des Datenvolumens setzt voraus, dass die Szene in Würfel zerlegt wird und jeder Prozessor den Inhalt eines Würfels zeichnet. Von dieser Annahme ausgehend, wird eine Abschätzung der Größe eines solchen Würfels in der Bildebene durchgeführt. Neumann kommt zu dem Ergebnis, dass jeder Prozessor etwa $An^{-2/3}$ Pixel berechnen muss. Da die Aufteilung der Szene in ein Gitter nur für statische Szenen oder für Volumendatensätze praktikabel ist und dies eine Einschränkung bei der Aufteilung der Szene darstellt, wird für den Vergleich der verschiedenen Verfahren zur Bildkomposition davon ausgegangen, dass jeder Prozessor in den kompletten Bildbereich zeichnet und somit exakt A Pixel für die Bildkomposition herangezogen werden müssen.

Die Summe der zu übertragenen Pixel ist auch in diesem Verfahren identisch mit der zentralen Bildkomposition. Es werden jedoch mehr Daten parallel übertragen als bei der binären Bildkomposition. Dies liegt daran, dass zu jedem Zeitpunkt während der Bildkomposition alle Prozessoren beschäftigt sind. Ein Nachteil dieses Verfahrens besteht darin, dass das fertige Bild auf alle Prozessoren verteilt ist.

6.2.4 Binary-Swap

Das „Binary-Swap"-Verfahren wurde von Ma, Painter, Hansen und Krogh [MPH94] entwickelt. Bei diesem Verfahren wird das fertige Bild in mehreren Iterationsschritten berechnet. Es werden Prozessorpaare gebildet, und der Bildbereich wird in zwei Hälften aufgeteilt. Jedem Prozessor wird eine Hälfte zugewiesen. Die beiden Prozessoren senden sich gegenseitig die zugewiesenen Hälften ihrer Bilder und führen eine Bildkomposition durch. Anschließend beginnt der nächste Iterationsschritt. Es werden Paare von Prozessoren mit gleichem Bildbereich gebildet. Dieser Bereich wird wieder halbiert, und hierfür wird eine Bildkomposition durchgeführt. Bei jedem Iterationsschritt werden die Bildbereiche kleiner. Das Verfahren ist nach $log_2 n$ Schritten beendet. Wie bei Direct-Send besitzt jeder Prozessor einen Teil des fertigen Bildes. Die folgende Abschätzung zeigt, dass sowohl die aufsummierte als auch die maximale sequentiell übertragene Datenmenge für Direct-Send und Binary-Swap identisch sind.

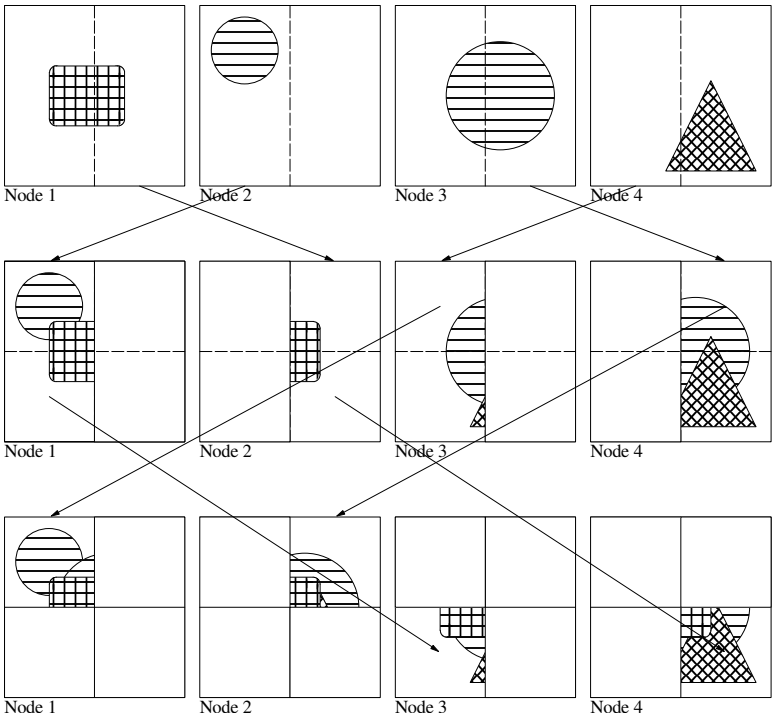

Abbildung 6-5: Binary-Swap

Der Vorteil dieses Verfahrens kommt dann zum Tragen, wenn jeder Prozessor nur einen kleinen Teil des Bildes füllt. Ist dies der Fall, wird in den ersten Iterationsschritten die Bildkomposition für große Bereiche durchgeführt. Wenn diese Bereiche allerdings nur wenig gefüllt sind, müssen mit einem Sort-Last-Sparse-Ansatz nur kleine Teile übertragen werden. Mit jedem Iterationsschritt werden mehr Bildpunkte gesetzt, allerdings werden gleichzeitig auch die Bereiche kleiner. In [MPH94] wird als weiterer Vorteil herausgestellt, dass die Anzahl der Nachrichten geringer ist als bei Direct-Send. Es wird zwar die gleiche Datenmenge übertragen, jedoch sind die Nachrichten größer, wodurch sich meist ein höherer Durchsatz erreichen lässt.

$$d_{seq} = A\sum_{i=1}^{\log_2 n} 2^{-i} = A\left(1-\frac{1}{n}\right)$$
$$m_{seq} = \log_2 n$$
$$d_{sum} = n \cdot d_{seq}$$

6.2.5 Distributed-Snooping

Michael Cox und Pat Hanrahan beschreiben in [CoH94] ein Verfahren, das an Cache-Mechanismen für Multiprozessor-Systeme mit Shared Memory angelehnt ist.

Das Verfahren beginnt damit, dass ein Prozessor alle Pixel, die in seinem Bildspeicher gezeichnet wurden, auf einem Datenbus sendet, an dem alle anderen Prozessoren angeschlossen sind. Die anderen Prozessoren lesen die Pixel vom Datenbus und vergleichen die Z-Werte mit den lokal gespeicherten Werten. Werden lokale Pixel überdeckt, so werden sie als nicht aktiv

gekennzeichnet. Hat der erste Prozessor alle aktiven Pixel gesendet, sendet Prozessor 2 alle aktiven Pixel auf den Datenbus. Das Verfahren ist abgeschlossen, wenn der letzte Prozessor seine aktiven Pixel gesendet hat.

Die Idee hinter diesem Algorithmus beruht darauf, dass mit jedem Schritt die Anzahl der aktiven Pixel sinkt. Im schlechtesten Fall, d.h., wenn die Bilder der Prozessoren in Schichten von hinten nach vorne angeordnet sind, werden von jedem Prozessor alle Pixel auf den Bus geschrieben. Cox und Hanrahan führen jedoch aus, dass dies in realen Situationen sehr unwahrscheinlich ist.

Nimmt man an, dass die Entfernung der Pixel zum Betrachter zufällig verteilt ist, dann ist ein Pixel an der Stelle x, y beim ersten Prozessor mit einer Wahrscheinlichkeit von 1 sichtbar. Beim zweiten Prozessor besteht eine Wahrscheinlichkeit von 1/2, dass dieses Pixel vor dem Pixel des ersten Prozessors liegt. Beim dritten Prozessor beträgt die Wahrscheinlichkeit nur noch 1/3 usw.

$$d_{seq} = A\sum_{i=1}^{i<\delta}\frac{1}{i}$$
$$m_{seq} = n$$
$$d_{sum} = d_{seq}$$

Die von Cox und Hanraham in [CoH92,CoH94] beschriebene Formel zur Abschätzung der zu übertragenden Datenmenge ist abhängig von der Tiefenkomplexität der Szene. Wird eine konstante Verteilung der Tiefenkomplexität über den Bildbereich angenommen, so kann die Summe der zu übertragenden Pixel mit der oben aufgeführten Formel abgeschätzt werden.

Für die Implementierung des Distributed-Snooping-Verfahrens ist eine Bus-Architektur erforderlich. Cox und Hanraham gehen davon aus, dass hierfür die vorhandene Bus-Architektur in Multiprozessor-Systemen mit Shared Memory verwendet werden kann.

Die Datenmenge, die bei der Komposition eines Bildes übertragen werden muss, ist bei diesem Verfahren nicht konstant. Sie ist abhängig von der Tiefenkomplexität und der Aufteilung der Szene auf die verschiedenen Prozessoren. Auch unter optimalen Voraussetzungen benötigt der erste Iterationsschritt mehr Zeit als die komplette Bildkomposition auf der Basis von Direct-Send oder Binary-Swap.

Für das Verfahren spricht die sehr einfache Kommunikationsstruktur. Betrachtet man nur die Summe der übertragenen Daten, so ist das Distributed-Snooping-Verfahren effizienter als alle anderen hier vorgestellten Arten der Bildkomposition.

6.2.6 Pipeline-Bildkomposition

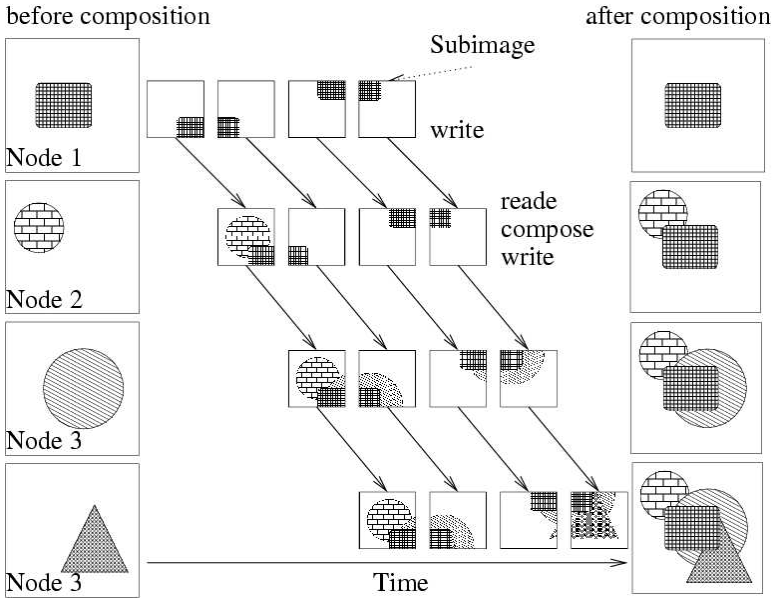

Abbildung 6-6: Pipeline-Bildkomposition

Bei der Pipeline-Bildkomposition sind alle Prozessoren hintereinander angeordnet. Der erste Prozessor sendet sein Bild an den zweiten. Dieser führt eine Bildkomposition durch und schickt das Ergebnis weiter an den nächsten Prozessor. Das Verfahren ist beendet, wenn der letzte Prozessor in der Pipeline mit dem Ergebnis seines Vorgängers eine Bildkomposition durchgeführt hat. Dieses Verfahren wurde bisher hauptsächlich für Hardware-Entwicklungen wie beispielsweise FixelFlow [MEP92], Sepia [MHS99,HeM99], Metabuffer [BBF00] oder Lightning-2 [SEP01] eingesetzt. Da jeder Prozessor nur mit jeweils zwei weiteren Prozessoren kommunizieren muss und das Verfahren sehr einfach ist, lässt es sich mit relativ geringem Aufwand hardwaretechnisch realisieren.

Auf den ersten Blick erscheint die Pipeline-Bildkomposition als sequentieller Prozess. Eine parallele Verarbeitung wird erst dann möglich, wenn die Bilder zwischen den Prozessoren in kleinen Einheiten versendet werden. Die Auslastung der Pipeline wird besser, je kleiner die übertragenen Bildteile sind. Hierbei ist jedoch zu beachten, dass viele Kommunikationssysteme wie beispielsweise Ethernet oder Myrinet [BCF95] bei sehr kurzen Nachrichten ineffizient sind. Es muss daher für die Größe der Bildteile ein Kompromiss gefunden werden, der von der eingesetzten Hardware abhängig ist.

In die folgende Abschätzung der übertragenen Datenvolumen fließt die Größe der übertragenen Bildteile als A_{tile} ein.

$$d_{seq} = A + (n-1)A_{tile}$$

$$m_{seq} = \frac{A}{A_{tile}} + n - 1$$

$$d_{sum} = A(n-1)$$

Die Pipeline-Bildkomposition erfordert eine einfache Netzwerktopologie, bei der jeweils zwei Prozessoren direkt miteinander verbunden sind. Alternativ können auch Mehrpunktverbindungen verwendet werden, falls diese über einen Switch realisiert sind. Eine Voraussetzung ist jedoch, dass der Switch eine bidirektionale, parallele und unabhängige Kommunikation zwischen verschiedenen Prozessoren ermöglicht. Einige Switche verbinden die einzelnen Kommunikationswege über einen gemeinsamen Datenbus. Ist die Bandbreite dieses Datenbusses kleiner als die Summe der Bandbreiten der Einzelverbindungen, dann kann der Switch zum Flaschenhals bei der Pipeline-Bildkomposition werden.

Im Gegensatz zu Direct-Send und Binary-Swap liegt das fertige Bild bei der Pipeline-Bildkomposition an einem Prozessor vor. Dies erleichtert die Darstellung auf Standard-Hardware.

6.2.7 Parallele Pipeline-Bildkomposition

In [LRN96] beschreiben Tong-Yee Lee, Raghavendra und Nicholas eine leicht modifizierte Variante der Pipeline-Bildkomposition, bei der am Ende der Verarbeitung jeder Prozessor einen Teil des fertigen Bildes besitzt. Abbildung 6-7 zeigt, wie die einzelnen Bildteile an die nachfolgenden Prozessoren verschickt werden. Im Gegensatz zum oben beschriebenen Ansatz sind hierbei die Prozessoren in einem Ring miteinander verbunden. Dieses Verfahren wird häufig auch als „Parallel Pipeline Composition“ bezeichnet.

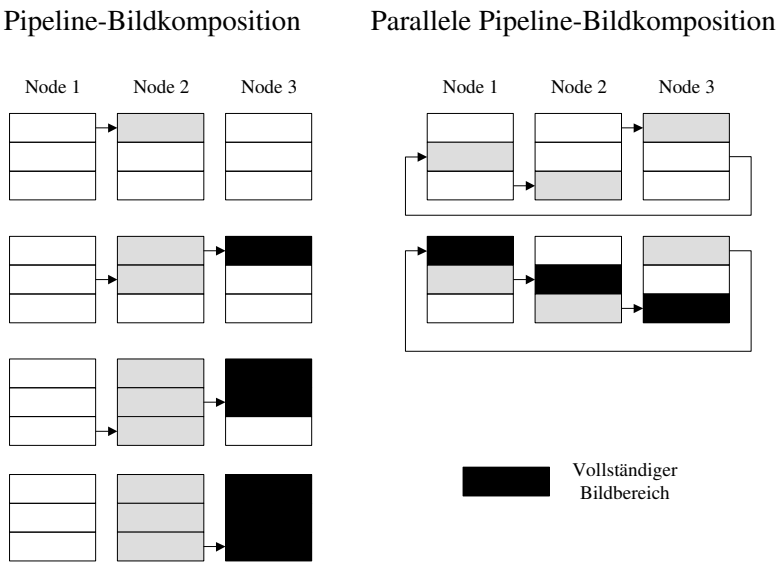

Abbildung 6-7: Unterschiedliche Pipeline-Verfahren zur Bildkomposition

$$d_{seq} = A\left(1 - \frac{1}{n}\right)$$
$$m_{seq} = n - 1$$
$$d_{sum} = A(n-1)$$

Wenn das fertige Bild nicht an einem Prozessor vorliegen muss, ist die parallele Pipeline-Bildkomposition schneller als das oben beschriebene einfache Pipeline-Verfahren. In [LRN96] und [MiC98] wird eine mehrstufige parallele Pipeline beschrieben. Tong-Yee Lee u.a. beschreiben eine Bildkomposition für zweidimensional vernetzte Prozessoren, bei der zuerst eine zeilenweise und anschließend eine spaltenweise Komposition durchgeführt wird. Mitra und Chiuhe zeigen in [MiC98], dass die Binary-Swap-Bildkomposition mit einer mehrstufigen parallelen Pipeline realisiert werden kann. Es wird beschrieben, dass hierbei *O(n)* Schritte zur Bildkomposition erforderlich sind. Aus diesem Grund wird versucht, durch ein mehrstufiges Konzept den Einfluss von *n* zu reduzieren. Es wird dabei jedoch übersehen, dass mit steigender Anzahl der Schritte auch der Aufwand pro Schritt sinkt. Der Einfluss von *n* ist daher sehr gering. Dies ist auch aus den in [MiC98] präsentierten Ergebnissen ersichtlich.

6.2.8 Ergebnis der Analyse

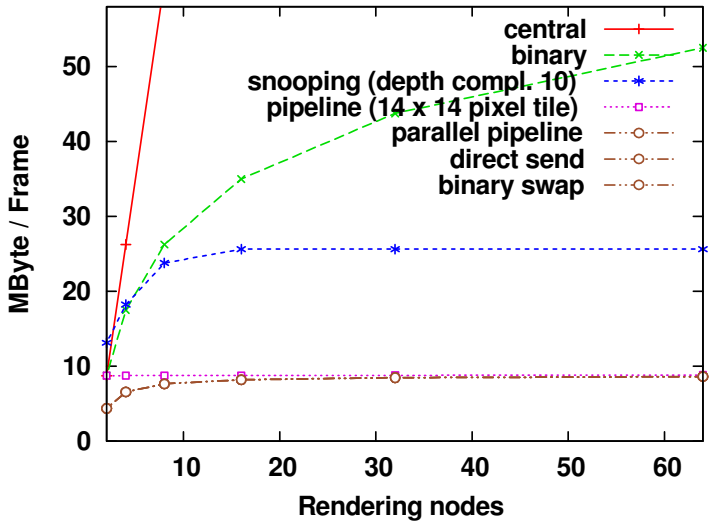

Abbildung 6-8: Vergleich der Verfahren zur Bildkomposition

Im nun folgenden Abschnitt werden die oben beschriebenen Verfahren bewertet und miteinander verglichen. Für die zu erwartende Performanz bei der Bildkomposition spielt die zu übertragende Datenmenge eine entscheidende Rolle. Allerdings ist nicht die Summe der Daten entscheidend, sondern das Maximum aller parallel übertragenen Datenströme. Abbildung 6-8 zeigt dies für alle untersuchten Verfahren. Es ist zu beachten, dass für Direct-Send, Binary-Swap und für die parallele Pipeline-Bildkomposition die gleiche Datenmenge übertragen wird. Auf die unterschiedliche Nachrichtengröße dieser drei Verfahren wird später noch eingegangen. Es ist offensichtlich, dass

die zentrale Bildkomposition für eine effiziente Bildkomposition nicht geeignet ist. Auch die binäre Komposition ist sehr stark von der Anzahl der Prozessoren abhängig und wird deshalb heute kaum noch eingesetzt. Aus diesem Grund werden diese beiden Verfahren im weiteren Vergleich nicht mehr berücksichtigt. Unter dem Aspekt der Datenübertragung sind Direct-Send, Binary-Swap und die parallele Pipeline die effizientesten Verfahren.

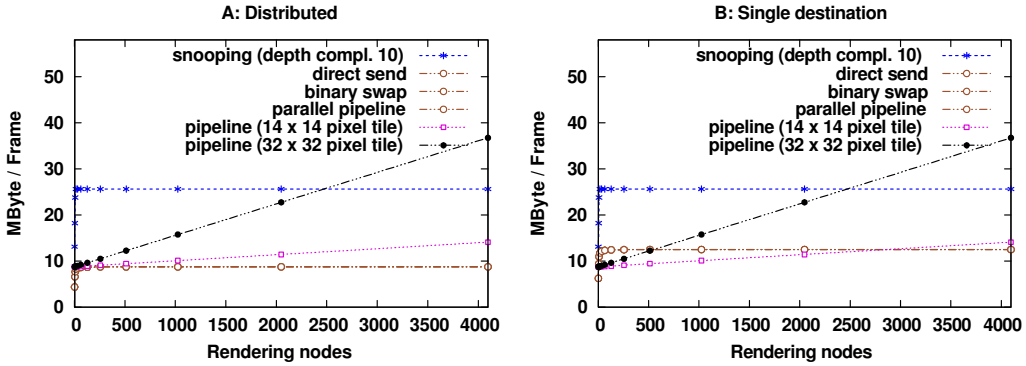

Abbildung 6-9: Verteilte und zentrale Bildausgabe

Bei einigen der oben beschriebenen Verfahren liegt das fertige Bild verteilt auf mehrere Prozessoren vor. In einigen Anwendungen kann es jedoch erforderlich sein, dass das Bild an einem einzigen Prozessor vorliegt. Dies ist dann der Fall, wenn der Prozessor ein Cluster-Rechner ist, an dessen Grafikkarte eine Projektion angeschlossen ist. Liegen die Bilder verteilt über alle Prozessoren vor, muss das Bild in einem zusätzlichen Arbeitsschritt zusammengesetzt werden. Hierfür ist folgender Kommunikationsaufwand erforderlich.

$$d = A\left(1 - \frac{1}{n}\right)$$

Die Datenübertragung kann hierbei nicht parallelisiert werden, da ein einziger Prozessor die Ergebnisse aller anderen empfängt. Abbildung 6-9 zeigt den Kommunikationsaufwand, für den Fall, dass das Ergebnis an einem Prozessor vorliegen muss. Bis zu einer Anzahl von etwa 500 parallel arbeitenden Prozessoren ist das einfache Pipeline-Verfahren effizienter als Binary-Swap, Direct-Send und die parallele Pipeline. Die Effizienz des einfachen Pipeline-Verfahrens ist abhängig von der Größe der übertragenen Teilbilder. Für diesen Vergleich wurde eine Größe der Teilbilder von 14 x 14 Bildpunkten angenommen, womit man auf eine für Ethernet optimale Nachrichtengröße von etwa 1500 Bytes kommt.

Auch unter der Annahme, dass das fertige Bild an einem Prozessor vorliegen muss, verursacht das Distributed-Snooping-Verfahren ein wesentlich größeres Datenvolumen als die vier restlichen verbleibenden Verfahren. Aufgrund der einfachen Kommunikationsstruktur, die diesem Verfahren zugrunde liegt, kann es durchaus sinnvolle Anwendungen geben. Aus Effizienzgründen ist es jedoch den restlichen noch verbleibenden Verfahren unterlegen.

Aus Abbildung 6-9 ist abzulesen, dass das einfache Pipeline-Verfahren ein geringeres Datenvolumen für die Netzwerkübertragung verursacht. Erst ab einer sehr hohen Zahl parallel arbeitender Prozessoren ist es effizienter, Binary-Swap, Direct-Send oder die parallele Pipeline zu verwenden, bei denen das fertige Bild aus Teilbildern verschiedener Prozessoren zusammengesetzt werden muss.

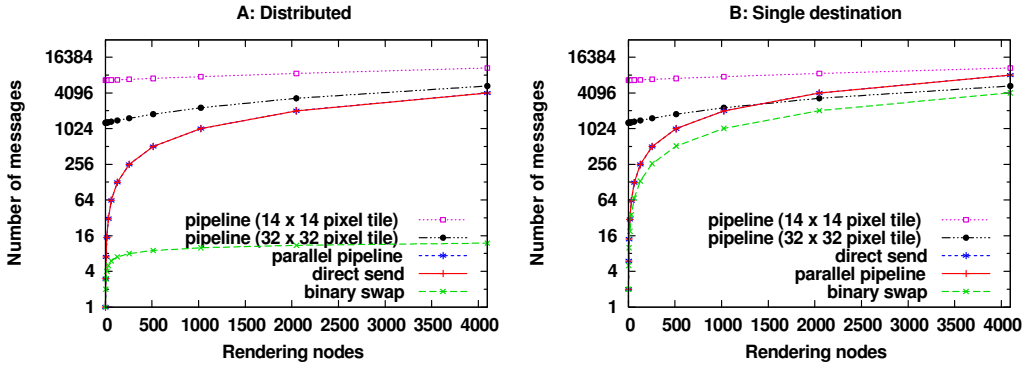

Abbildung 6-10: Anzahl der versendeten Nachrichten

Neben dem reinen Datenvolumen ist für den Vergleich der Verfahren auch die Größe der zu sendenden Nachrichten wichtig. Die meisten Systeme zur Nachrichtenübertragung benötigen für jede Nachricht einen fixen Aufwand. Aus Abbildung 6-10 kann die Anzahl der versendeten Nachrichten für ein verteiltes und für ein zentrales Ergebnisbild abgelesen werden. Es ist deutlich zu sehen, dass für die Binary-Swap-Komposition die wenigsten Nachrichten verschickt werden müssen. Da, wie in Abbildung 6-8 abzulesen ist, Binary-Swap das gleiche Datenvolumen verursacht wie Direct-Send und die parallele Pipeline, bedeutet dies, dass wesentlich längere Nachrichten versendet werden. Berücksichtigt man die fixen Kosten für die Übertragung einer Nachricht, kann mit Binary-Swap ein höherer Datendurchsatz erreicht werden.

Betrachtet man die Untersuchungsergebnisse, so lassen sich folgende Entscheidungskriterien für oder gegen die einzelnen Verfahren aufführen.

- **Zentrale Bildkomposition**: Dieses Verfahren ist für die Bildkomposition ungeeignet. Da bei diesem Verfahren die Bildkomposition sequentiell durchgeführt wird, ergeben sich mit einer steigenden Zahl von Prozessoren sehr lange Verarbeitungszeiten.

- **Binäre Komposition**: Auch dieses Verfahren ist nicht für eine effiziente Implementierung geeignet. Die übertragene Datenmenge ist im Vergleich zur zentralen Bildkomposition zwar geringer, aber immer noch mit $O(log(n))$ von der Anzahl der Prozessoren abhängig.

- **Direct-Send**: Effizienter als die binäre Komposition, allerdings ist die Anzahl der versendeten Nachrichten größer als bei Binary-Swap. Aus diesem Grund ist Binary-Swap diesem Verfahren überlegen. Bei Direct-Send liegen die fertigen Bilder nicht an einem Prozessor vor.

- **Binary-Swap**: Falls das fertige Bild nicht an einem einzigen Prozessor vorliegen muss, ist Binary-Swap das effizienteste der hier vorgestellten Verfahren. Die Datenmenge wird in wenigen großen Nachrichtenblöcken versendet. Damit lassen sich auch in Netzwerken mit hoher Latenz gute Ergebnisse erzielen. Bei Verwendung von Sort-Last-Sparse können mit Binary-Swap große Datenbereiche ausgeklammert werden. Da jeder Prozessor mit jedem anderen kommunizieren muss, ist ein komplexes und leistungsfähiges Kommunikationsnetzwerk erforderlich.

- **Distributed-Snooping**: Das Verfahren ist von der Tiefenkomplexität der Szene abhängig. Die Anzahl der Prozessoren spielt dagegen für die Verarbeitungszeit keine Rolle. Es ist nicht so effizient wie Binary-Swap, allerdings ist als Kommunikationsebene nur ein einfacher Datenbus erforderlich. Eine solche Architektur lässt sich auch für einen sehr hohen Parallelisierungsgrad kostengünstig realisieren.

- **Pipeline**: Das Verfahren ist einfach zu realisieren und erfordert lediglich eine einfache Vernetzung. Falls das fertige Bild an einem einzigen Prozessor vorliegen soll, ist dieses Verfahren auch bei mehreren hundert Prozessoren effizienter als Binary-Swap mit anschließendem Zusammensetzen des Bildes.

- **Parallele Pipeline**: Der Kommunikationsaufwand ist bei diesem Verfahren mit Direct-Send und Binary-Swap vergleichbar. Es verursacht jedoch, wie Direct-Send, mehr Nachrichten als Binary-Swap. Im Gegensatz zu Direct-Send ist die erforderliche Netzwerk-Infrastruktur jedoch einfacher.

In diesem Abschnitt wurden die z.Z. gängigen Verfahren zur Bildkomposition aufgeführt und bewertet. Der nun folgende Abschnitt beschreibt die im Rahmen dieser Arbeit entwickelte „Sorted-Pipeline Composition“. Im Anschluss daran folgt ein Vergleich mit bereits existierenden Verfahren.

6.3 Die Sorted-Pipeline-Bildkomposition

Die im vorangegangen Abschnitt beschriebene Untersuchung bestehender Verfahren zur Bildkomposition hat gezeigt, dass selbst mit den effizientesten Verfahren bei einer Standard-Bildschirmauflösung mehr als eine Million Bildpunkte mit Tiefeninformationen von jedem parallel arbeitenden Knoten gelesen und versendet werden müssen. Um bei diesem Datenvolumen eine interaktive Bildwiederholrate zu erreichen, wäre eine 10-Gigabit-Verbindung zwischen allen Knoten des Clusters erforderlich. Dies ist zwar technisch bereits möglich, hierbei würden die Kosten für das Netzwerk die Kosten für den Cluster um ein Vielfaches übersteigen. Es ist daher notwendig, das Datenvolumen drastisch zu reduzieren.

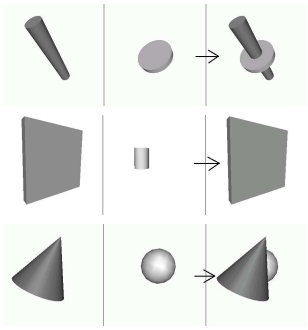

Abbildung 6-11: Varianten der Bildkomposition

Eine häufig eingesetzte Technik besteht darin, in jedem Teilbild nur die Bereiche zu betrachten, die sichtbare Teile der Geometrie enthalten. Wird eine Szene auf sehr viele parallele CPUs verteilt, überdecken die Teilszenen häufig nur einen kleinen Teil des Bildbereichs. Für die Bildkomposition ist nur dieser Bereich interessant. Der Bereich, in dem eine Szene auf dem Bildschirm abgebildet wird, lässt sich näherungsweise schnell anhand des Hüllvolumens bestimmen. Projiziert man das Volumen, das eine Szene umschließt, auf den Bildschirm, erhält man den Bereich, der die darzustellende Szene komplett enthält. Alle Bildpunkte, die außerhalb dieses Bereiches liegen, enthalten nur Farbwerte des Bildhintergrundes. Wenn bei der Aufteilung der Szenen darauf geachtet wird, dass alle Teilszenen aus räumlich zusammenliegenden Objekten bestehen, kann man das Datenvolumen bereits stark reduzieren. Diese Technik wird meist als Sort-Last-Sparse bezeichnet. Im Gegensatz hierzu wird die Bildkomposition der vollständigen Bilderebene als Sort-Last-Full bezeichnet.

Das Kompositionsverfahren, welches im Rahmen dieser Arbeit entwickelt wurde, basiert auf Sort-Last-Sparse. Es werden darüber hinaus jedoch Techniken entwickelt, die das Datenvolumen weiter reduzieren. Die Hauptidee besteht darin, vor der eigentlichen Bildkomposition die einzelnen Teilbilder zu analysieren, um unnötige Datenübertragungen zu vermeiden. Wenn beispielsweise bekannt ist, dass ein Teilbild komplett hinter einem anderen Teilbild liegt, dann ist es nicht mehr notwendig, die Tiefeninformation zu übertragen. Ist zusätzlich bekannt, dass ein Teilbild ein anderes vollständig überdeckt, kann das überdeckte Teilbild bereits vor der Bildkomposition verworfen werden. Abbildung 6-11 zeigt Situationen, die bei der Bildkomposition auftreten können. Eine Bildkomposition, bei der für jeden Bildpunkt die exakte Tiefe berücksichtigt werden muss, ist nicht immer erforderlich.

Das hier vorgestellte Verfahren wurde mit dem Ziel entwickelt, die Bildberechnung einer 3D-Echtzeitanwendung zu beschleunigen. Das Ergebnis der Bildkomposition muss hierbei an einem Display angezeigt werden. Für die Bildkomposition nach dem Binary-Swap-Prinzip bedeutet dies, dass nach der Komposition die Einzelergebnisse an das Display versendet werden müssen. Unter dieser Randbedingung bietet die Komposition in einer Pipeline die besten Voraussetzungen. In kleinen bis mittleren Clustern wird bei der Pipeline-Komposition das geringste Datenvolumen übertragen. Die Pipeline-Komposition liefert das Ergebnis an einem Knoten und hat zusätzlich den Vorteil, dass eine beliebige Anzahl von Knoten verwendet werden kann. Im Gegensatz hierzu muss bei Binary-Swap eine Zweierpotenz von Knoten vorhanden sein. Dies kann mit steigender Zahl

von Knoten sehr ungünstig sein. Im Fraunhofer Institut für Grafische Datenverarbeitung in Darmstadt stehen beispielsweise mehrere PC-Cluster mit zusammen 90 Rechnern zur Verfügung. Mit einer Binary-Swap-Komposition können hiervon lediglich 64 Rechner verwendet werden.

6.3.1 Vorverarbeitung zur optimierten Bildkomposition

Die Anzahl der Bildpunkte, die bei der Bildkomposition berücksichtigt werden muss, kann wesentlich reduziert werden, wenn bei der Bildkomposition nicht nur lokale Informationen wie beispielsweise das Hüllvolumen, sondern auch globale Informationen wie beispielsweise maximale und minimale Bildtiefe eines Bildbereichs verwendet werden. Bei einer Pipeline-Komposition sind alle Knoten nacheinander angeordnet. Auf der Basis globaler Informationen lassen sich folgende Fälle unterscheiden:

- Alle Bildpunkte eines Teilbildes liegen hinter den Bildpunkten der Knoten, die in der Pipeline noch folgen. In diesem Fall muss keine Tiefe übertragen werden, die nachfolgenden Knoten können die Bilddaten mit einem konstanten maximalen Tiefenwert in den Bildspeicher kopieren. Hierdurch werden pro Bildpunkt 4 Bytes eingespart.

- Alle Bildpunkte werden von Teilbildern der nachfolgenden Knoten überdeckt. In diesem Fall können diese Bildpunkte verworfen werden. Sie sind im fertig zusammengesetzten Bild nicht sichtbar. In komplexen Modellen tritt dieser Fall häufig auf. Insbesondere in Modellen von Gebäuden ist eine große Anzahl von Bildpunkten nicht sichtbar, da sie von Wänden verdeckt werden.

- Teile der Bildpunkte liegen vor und andere hinter Teilbildern der nachfolgenden Knoten. In diesem Fall muss neben der Farbe auch die Tiefe der Punkte übertragen werden.

Um diese Entscheidungen treffen zu können, müssen lediglich die minimalen und maximalen Tiefenwerte aller Teilbilder untersucht werden. Ermittelt man die Tiefenwerte für den kompletten Bildbereich, sind für die Analyse nur sehr wenige Daten zu übertragen. Es ist dann jedoch sehr selten der Fall, dass ein Teilbild ein anderes vollständig überdeckt oder vollständig hinter einem anderen Teilbild liegt. Die Wahrscheinlichkeit steigt jedoch, wenn man kleinere Bildbereiche untersucht. Im Extremfall könnte man jeden Bildpunkt betrachten und somit jede überflüssige Datenübertragung vermeiden. Allerdings müssen dann für die Analyse mehr Daten übertragen werden als für die eigentliche Bildkomposition. Es muss daher ein Kompromiss gefunden werden. Untersuchungen mit unterschiedlichen Kachelgrößen haben gezeigt, dass mit einer Größe von 44 x 44 Pixeln ein gutes Verhältnis zwischen Analyseaufwand und Datenreduktion zu erreichen ist.

Für jede Kachel sendet jeder Knoten die folgenden Informationen an eine zentrale Stelle:

- Enthält die Kachel lediglich Bildpunkte des Bildhintergrunds?
- Welche maximale Entfernung haben die Punkte der Kachel vom Betrachter?

- Welche minimale Entfernung haben die Punkte der Kachel zum Betrachter?
- Enthält die Kachel Bildpunkte des Hintergrunds?

Bildpunkte des Bildhintergrunds lassen sich in OpenGL sehr einfach und schnell ermitteln. Der Hintergrund wird i.d.R. mit dem größten Z-Wert in den Z-Buffer geschrieben. Bei einer Bittiefe von n ist die Tiefe des Hintergrunds 2^{n-1}.

Für die Analyse der Bildbereiche wird der vollständige Bildspeicher mit allen Farb- und Tiefenwerten aus der Grafikkarte ausgelesen und in den Hauptspeicher kopiert. Anschließend werden per Software alle Bildpunkte analysiert und die maximalen und minimalen Tiefenwerte ermittelt. Auf vielen Grafikkarten ist das Auslesen der Tiefeninformation sehr zeitaufwendig. Beispielsweise lassen sich die Tiefenwerte auf einer ATI-Radon-9600-Grafikkarte mit einer Auflösung von 1024 x 768 Bildpunkten nur etwa zweimal pro Sekunde auslesen. Das macht den Einsatz dieser Grafikkarte für alle Sort-Last basierten Verfahren ungeeignet. Die im Folgenden aufgeführten Ergebnisse dieser Arbeit wurden auf einem Cluster mit NVidia-Grafikkarten erzielt. Diese Grafikkarten erlauben es, etwa 30 mal pro Sekunde sowohl den Farb- als auch den Tiefenspeicher der Grafikkarte auszulesen.

6.3.2 Sortierung der Kompositions-Pipeline

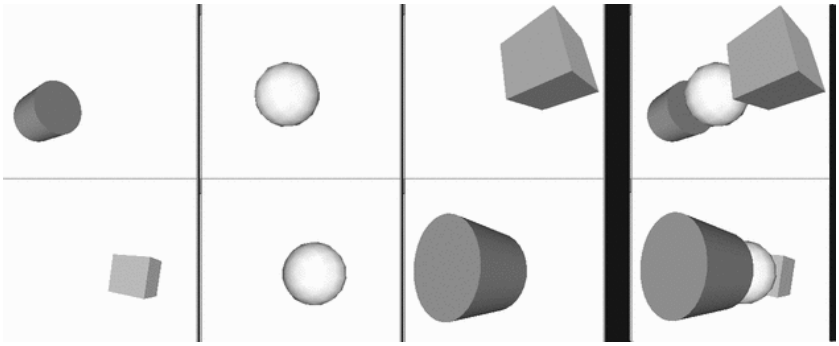

Abbildung 6-12: Sortierung der Pipeline erfolgt von hinten nach vorne

Die Wahrscheinlichkeit, mit der in einer Pipeline eine Datenübertragung vermieden werden kann, hängt wesentlich von der räumlichen Position der Teilszenen in den Knoten der Pipeline ab. Sind die Teilszenen im ungünstigsten Fall von vorne nach hinten sortiert, dann ist durch die oben aufgeführten Regeln keine Datenreduktion zu erwarten. Im Gegensatz dazu würde eine Sortierung von hinten nach vorne wesentlich häufiger eine teilweise oder vollständige Überdeckung liefern. Um eine Sortierung der Objekte in der Reihenfolge der Pipeline zu erreichen, könnte man jedem Rechner je nach Blickpunkt des Betrachters einen anderen Teil der Szene zuweisen. In [NgZ00] „Image Layer Decomposition for Distributed Rendering on NOWs“ wird für jedes Bild ein Überdeckungsgraph aufgebaut. Mit Hilfe dieses Graphen wird jedem Rechner ein Teil der Szene zugeordnet. Das hat jedoch den Nachteil, dass ein Rechner von Bild zu Bild andere Teile der Szene berechnen muss. Hierbei müssen eventuell Texturen ausgelagert oder Display-Listen neu geladen

werden. Ein weiterer Nachteil besteht darin, dass auf jedem Rechner die komplette Szene geladen werden muss.

Damit jeder Rechner in jedem Bild die gleiche Teilszene berechnen und dennoch eine Sortierung der Objekte von hinten nach vorne erfolgen kann, wird im vorliegenden Verfahren die Reihenfolge der Rechner in der Bildkompositions-Pipeline in jedem Bild angepasst. Die Reihenfolge der Rechner in der Pipeline wird durch die aktuelle Position der Teilszene bestimmt. D.h., die Reihenfolge der Rechner ist nicht konstant, sondern sie kann sich von Bild zu Bild ändern. Hierdurch sind die Teilszenen in der Pipeline von hinten nach vorne sortiert, und jeder Rechner muss dennoch nur die gleiche Teilszene berechnen. Dadurch lassen sich mit dem vorliegenden Verfahren auch Szenen laden und darstellen, die aufgrund ihrer Größe nicht vollständig von einem einzelnen Rechner geladen werden können.

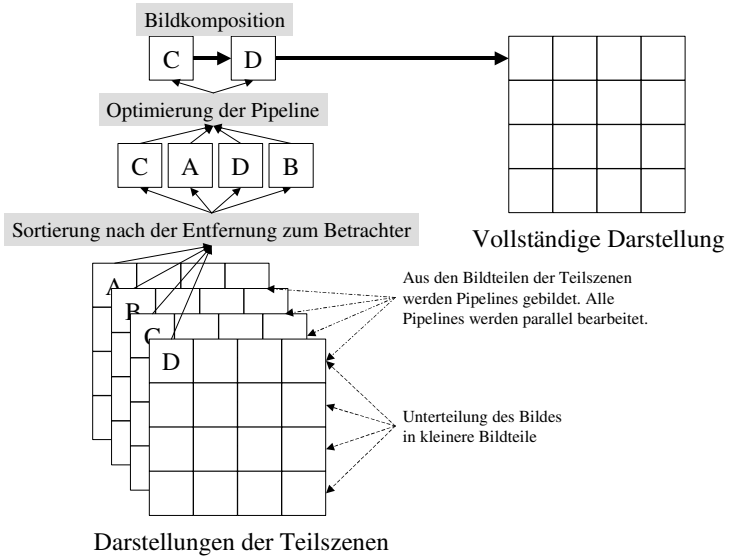

Abbildung 6-13: Schematischer Ablauf der Sorted-Pipeline-Bildkomposition

Führt man die Sortierung für den kompletten Bildbereich durch, wird man nur eine sehr grobe Sortierung von hinten nach vorne erhalten. Dies liegt daran, dass in realen Szenen eine gute räumliche Aufteilung nur schwer möglich ist. Beispielsweise kann die Karosserie eines Autos ein Objekt sein und das Lenkrad ein anderes. Eine sinnvolle globale Sortierung ist hierbei nicht möglich. Wie oben beschrieben, wird für die Analyse der Bildbereich in kleine Teilbereiche aufgeteilt. Im vorliegenden Verfahren wird jeder Teilbereich als eigenständige Pipeline betrachtet. Jede unterteilte Pipeline kann nun getrennt sortiert werden, d.h., die Komposition der einzelnen Teilbereiche erfolgt in unterschiedlichen Reihenfolgen. Die Sortierung ist hierbei wesentlich genauer. Im Falle des Lenkrads und der Karosserie könnte das bedeuten, dass in einem Teilbereich die Karosserie vor dem Lenkrad liegt und in einem anderen Bereich das Lenkrad einen Teil der Karosserie überdeckt.

6.3.3 Reduktion der Pipelinestufen

Aufgrund der eingesetzten Techniken wird das im Rahmen dieser Arbeit entwickelte Verfahren als „Sorted-Pipeline“-Komposition bezeichnet. Das Datenvolumen lässt sich bei Verwendung von Sort-Last-Sparse noch einmal wesentlich reduzieren. Eine weitere Möglichkeit, die Effizienz zu steigern besteht darin, die Länge der Pipeline zu reduzieren. Ein Problem für den Einsatz von Pipelines besteht darin, dass die Verzögerung mit der Länge der Pipeline zunimmt. Da beim Sorted-Pipeline-Verfahren sehr viele Einzelpipelines verwendet werden, ist es sehr unwahrscheinlich, dass jeder Knoten zu jedem Bildbereich Bilddaten liefert. Aufgrund der Analyseergebnisse können die Knoten aus den Pipelines entfernt werden, die im entsprechenden Bildbereich keine Teilszenen gezeichnet haben. Hiermit lässt sich die Länge der meisten Pipelines auf einen oder wenige Knoten reduzieren.

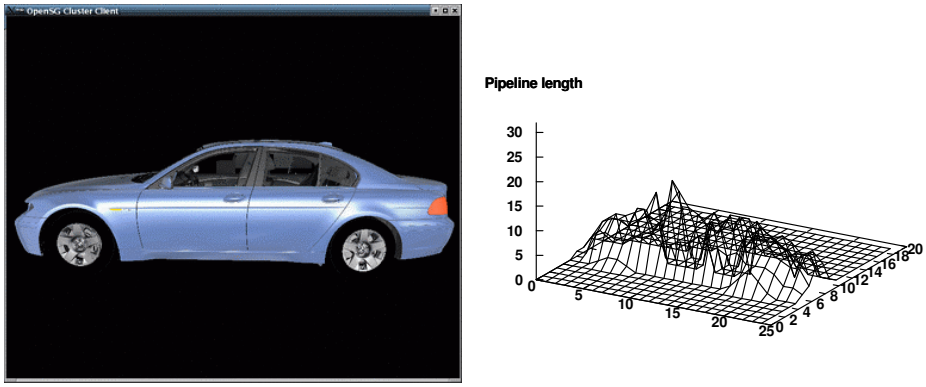

Abbildung 6-14: Länge der optimierten Pipelines für das BMW-Modell

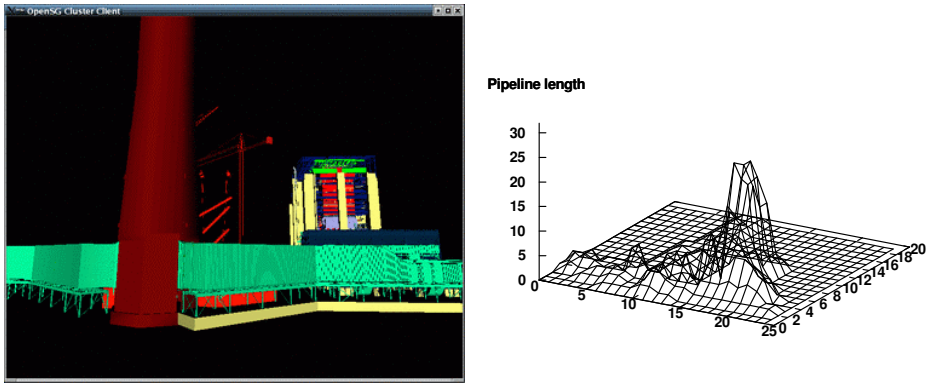

Abbildung 6-15: Länge der optimierten Pipelines für das Power-Plant-Modell

Abbildung 6-14 und Abbildung 6-15 zeigen deutlich, dass für einen großen Teil des Bildes sehr kurze Pipelines erforderlich sind. Auch in komplexen Szenen treten häufig Pipeline-Längen von 1 auf. Bei einer Länge von 1 ist keine Bildkomposition erforderlich. Sehr lange Pipelines treten an Stellen mit hoher Tiefenkomplexität auf. Im Falle des Power-Plant-Modells betrifft dies die Maschinenhalle. Dort sind Hunderte von Rohren auf sehr engem Raum konzentriert. Es ist auch zu erkennen, dass an den Stellen, an denen die Maschinenhalle durch Objekte im Bildvordergrund überdeckt wird, die Länge der Pipelines sehr kurz ist. Dies liegt daran, dass eine vollständige

Überdeckung erkannt wird und dadurch alle nachfolgenden Stufen der Pipeline ignoriert werden können. Eine genaue Analyse der Tiefenkomplexität in Zusammenhang mit der Sort-Last-parallelen Bildberechnung ist in [CoH92] zu finden. Hierbei wird jedoch nicht auf die Möglichkeit der Komplexitätsreduktion durch Verdeckung eingegangen.

6.3.4 Parallelisierung der Bildkomposition

In einem modernen PC können über die Netzwerkschnittstelle gleichzeitig Daten gesendet und geschrieben werden. Bei der Sorted-Pipeline-Bildkomposition müssen Bildteile gelesen, mit anderen Bildteilen verrechnet und anschließend weiter versendet werden. Eine optimale Performanz lässt sich nur dann erreichen, wenn die Verarbeitungsschritte parallelisiert werden. Hierzu wird das Lesen der Bilddaten von anderen Rechnern in einen eigenen parallel laufenden Prozess übernommen. Dieser Prozess kann die ankommenden Daten in der Geschwindigkeit lesen, wie sie über die Netzwerkschnittstelle angeliefert werden. Er ist dabei nicht von der Sendegeschwindigkeit und dem Berechnungsaufwand für die Komposition der Bilddaten abhängig. Parallel dazu werden lokal vorhandene Bildbereiche mit empfangenen Bildteilen zusammengefügt und an andere Rechner weitergeleitet.

Die Entkopplung des Sendens vom Schreiben hat auch den Vorteil, dass die Verarbeitung der Sub-Pipelines in beliebiger Reihenfolge erfolgen kann. Durch die Entkopplung ist es möglich, dass Rechner A auf ein Bildteil von Rechner B für die Sub-Pipeline 1 wartet, während gleichzeitig Rechner B ein Bildteil von Rechner A für eine andere Sub-Pipeline benötigt.

Eine Parallelisierung der Datenübertragung bringt nicht nur für die Sorted-Pipeline-Bildkomposition Vorteile, auch die Binary-Swap-Bildkomposition kann hiermit beschleunigt werden. Bei den am Ende dieses Kapitels aufgeführten Vergleichen zwischen Binary-Swap und Sorted-Pipeline wird eine parallele Datenübertragung bei beiden Verfahren verwendet.

6.3.5 Behandlung transparenter Flächen

Transparente Flächen lassen sich nicht ohne weiteres mit einem Sort-Last-Verfahren korrekt darstellen. Dies liegt daran, dass zur korrekten Darstellung zuerst alle nicht transparenten Objekte und anschließend alle transparenten Objekte von hinten nach vorne gezeichnet werden müssen. Für eine korrekte Darstellung sind daher die Entfernung eines Bildpunktes und die Farbe des Punktes erforderlich. Diese Informationen stehen jedoch bei einer optimierten Bildkomposition zu keiner Zeit zur Verfügung. Daher können transparente Flächen in den Teilszenen nicht korrekt gezeichnet werden und auch nicht im Ergebnis der Bildkomposition. In der Literatur wird dieses Problem für die Darstellung von Polygonen meist ignoriert [LRN96]. Bei der Darstellung von Volumendaten wird meist das Volumen so unterteilt, dass eine Bildkomposition von hinten nach vorne möglich ist [RaS99]. Polygonmodelle lassen sich zwar auch nach einem solchen Schema aufteilen, dies hat jedoch den Nachteil, dass ein Modell, das von einer Applikation geladen wird, vor der

Bildberechnung modifiziert werden muss. Insbesondere in interaktiven Anwendungen mit dynamischen Szenen ist eine Modifikation des Szenengraphen problematisch.

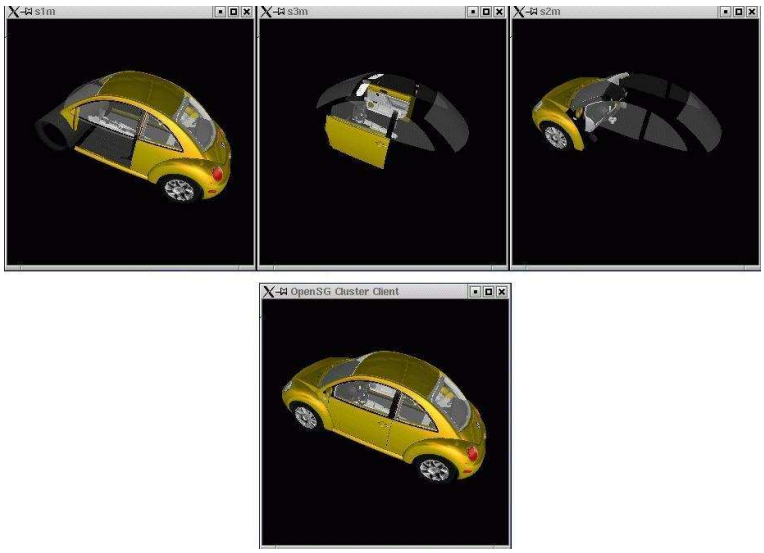

Abbildung 6-16: Parallele Bildberechnung mit transparenten Flächen

Im hier vorgestellten Verfahren wird daher eine Lösung präsentiert, die mit beliebigen Szenenunterteilungen funktioniert. Hierbei wird davon ausgegangen, dass in den meisten Darstellungen, die auf einem Polygonmodell basieren, die Anzahl der transparenten Flächen gering ist im Vergleich zu den nicht transparenten Flächen. Unter dieser Annahme ist es vertretbar, dass die Berechnung transparenter Flächen nicht parallelisiert wird. Jeder Rechner, der eine Teilszene berechnet, zeichnet alle sichtbaren transparenten Flächen in seine Teilszene.

Durch eine transparente Fläche kann in einer Teilszene ein falscher Teil der Szene überblendet werden. Transparente Objekte haben keinen Einfluss auf die Bildkomposition, da beim Zeichnen der Z-Wert des Tiefenpuffers nicht verändert wird. Dadurch wird bei der Bildkomposition nur der überblendete Bildpunkt in das Ergebnisbild übernommen, der dem Betrachter am nächsten ist. Durch diesen einfachen Mechanismus ist gewährleistet, dass nur die korrekten Überblendungen transparenter Objekte im Ergebnisbild zu sehen sind.

In Abbildung 6-16 ist beispielsweise die Darstellung transparenter Scheiben in einem Automodell zu sehen. Im oberen Teil sind die Ergebnisse von drei Rechnern vor der Bildkomposition zu sehen. Jeder der drei Rechner hat die transparenten Scheiben in seiner Teilszene gezeichnet. Auf dem dritten Bild der oberen Reihe ist zu sehen, wie die Heckscheibe den Bildhintergrund überlagert. Dies ist ein Fehler, denn die Heckscheibe soll korrekterweise den hinteren Innenbereich überdecken. Die korrekte Überblendung ist in Bild 1 zu sehen. Die korrekte Überblendung wird in das Ergebnisbild übernommen, da die sichtbaren Punkte des überdeckten Bereichs vor den Punkten in Bild 3 liegen. Im Ergebnisbild ist zu sehen, dass alle transparenten Flächen korrekt im Ergebnisbild zu sehen sind.

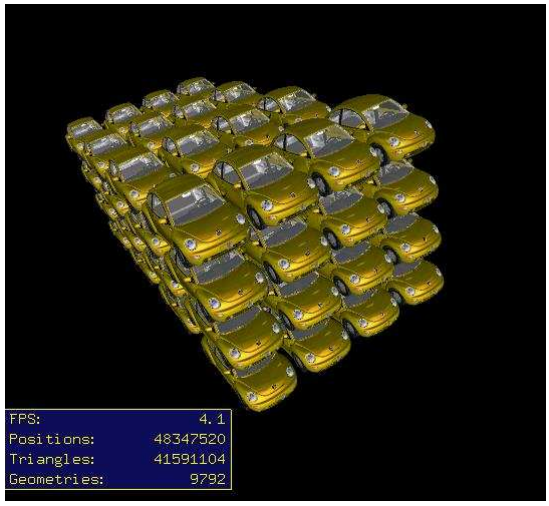

Abbildung 6-17: Bildberechnung auf 32 Rechnern mit transparenten Flächen

6.3.6 Optimierung der Teilbildgrößen

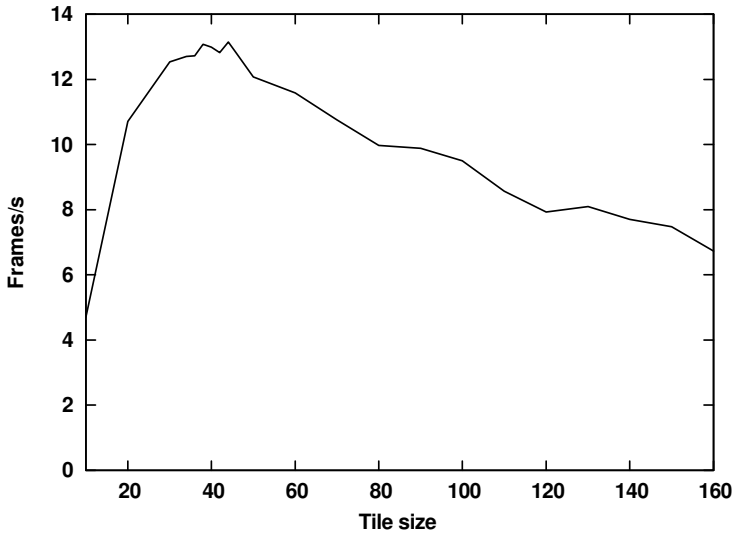

Abbildung 6-18: Verhältnis der Größe des Teilbereichs zur Bildwiederholrate

Die Größe der Teilbereiche, die bei der Sorted-Pipeline-Bildkomposition verwendet wird, hat einen großen Einfluss auf die Performanz. Kleine Bereiche führen zu einer sehr hohen Datenreduktion bei der Bildkomposition, jedoch müssen für die globale Sortierung viele Daten übertragen werden. Im Gegensatz dazu führen sehr große Bereiche dazu, dass wenige Daten bei der Bildkomposition optimiert werden können. Dafür müssen aber für große Bereiche nur geringe Datenmengen zur globalen Sortierung übertragen werden. Es muss daher ein Kompromiss gefunden werden.

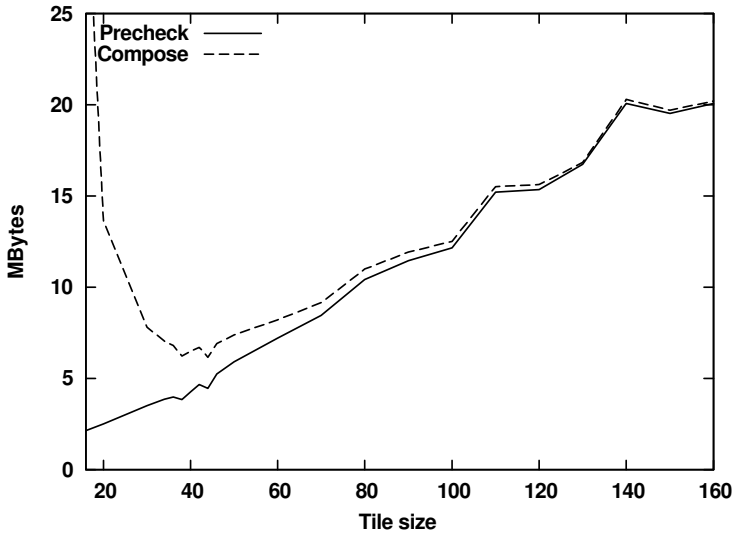

Abbildung 6-19: Datenmengen bei unterschiedlichen Tile-Größen

Abbildung 6-18 zeigt die Bildkomposition mit unterschiedlichen Größen der Teilbereiche. Es ist zu sehen, dass bei einer Kantenlänge der Teilbereiche von etwa 40 bis 60 die besten Ergebnisse erzielt werden. Für die in dieser Arbeit aufgeführten Performanzmessungen wurde eine Kantenlänge von 44 Pixeln festgelegt. Abbildung 6-19 zeigt die übertragenen Datenmengen bei unterschiedlichen Größen der Teilbereiche. Es ist deutlich zu sehen, dass mit sehr kleinen Bereichen nur wenige Daten für die Bildkomposition übertragen werden müssen. Allerdings erfordert die Übertragung aller Daten ein größeres Datenvolumen als bei der Komposition eingespart wird. Ein Optimum mit geringer Datenübertragung für die globale Sortierung und eine gute Optimierung der Bildkomposition ist bei einer Kantenlänge von 40 Pixeln zu erkennen.

6.3.7 Ergebnisse

Für den Test des vorgestellten Verfahrens wurden unterschiedliche Modelle ausgewählt. Die einzelnen Modelle besitzen sehr unterschiedliche Eigenschaften, so dass die Ergebnisse auf eine Vielzahl von Anwendungsfällen übertragen werden können. Für die ausgewählten Szenen wurde jeweils ein Animationspfad definiert, der sowohl das Modell im Überblick als auch bildschirmfüllende Nahaufnahmen beinhaltet.

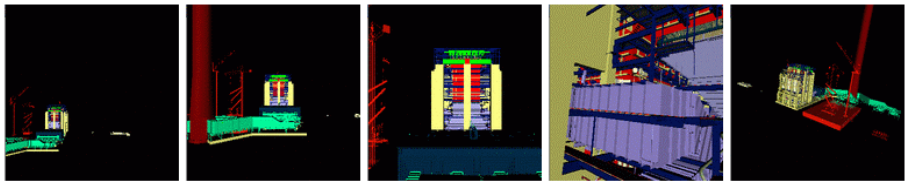

Abbildung 6-20: UNC Power Plant: 13 Millionen Dreiecke

Abbildung 6-21: BMW 7: Vier Millionen Dreiecke

Abbildung 6-22: Lucy: 28 Millionen Dreiecke (Stanford Scanning Repository)

Abbildung 6-23 David: 50 Millionen Dreiecke (Stanford Scanning Repository)

Die Komplexität des Power-Plant-Modells ist auf einen sehr kleinen Teil des Modells beschränkt. Der überwiegende Teil der 13 Millionen Polygone befindet sich in der Mitte des Maschinenraums. Diese Struktur ist der ungünstigste Fall für die Sorted-Pipeline-Bildkomposition, da hierdurch sehr lange Pipelines in diesen komplexen Bildbereichen entstehen. Die Polygone des BMW7-Modells sind über das gesamte Modell verteilt. Bereiche mit vielen Polygonen sind die Felgen und das Armaturenbrett. Große Teile der Geometrie werden meist durch die Karosserie verdeckt. Dies ist eine sehr gute Voraussetzung für den Verdeckungstest der Sorted-Pipeline-Bildkomposition. Das Lucy-Modell aus dem Scanning Repository der Stanford University besteht aus 28 Millionen Dreiecken. Die Dreiecke sind gleichmäßig über das Modell verteilt. Das Modell wurde mit Hilfe eines 3D-Scanners von einer Statue von Michelangelo erzeugt. Im Gegensatz zum BMW-Modell werden kaum Bereiche verdeckt. Da das Modell hohl ist, besteht eine sehr geringe Tiefenkomplexität. Das David-Modell ist das größte getestete Modell. Es besitzt etwa 50 Millionen Dreiecke und lässt sich auf einem einzelnen Rechner nur dann laden, wenn der verfügbare Hauptspeicher mindestens 1,5 GByte beträgt.

Im Folgenden wird das neue Sorted-Pipeline-Bildkompositionsverfahren mit dem sehr weit verbreiteten und effizienten Binary-Swap-Verfahren verglichen. Um einen aussagekräftigen Vergleich zu erreichen, werden alle Optimierungen, die in beiden Verfahren sinnvoll eingesetzt werden können, auch bei der Performanz-Messung berücksichtigt. In beiden Verfahren werden die folgenden Optimierungen vorgenommen.

- Beide Verfahren sind als Sort-Last-Sparse realisiert.
- Teile des Bildhintergrundes werden nicht übertragen.
- Das Lesen und Schreiben von Bildteilen ist parallelisiert.

Da die reine Ausführungsgeschwindigkeit durchaus von Details der Implementierung abhängen kann, wird zuerst untersucht, welche Reduzierung der zu übertragenden Datenmenge durch das Sorted-Pipeline-Verfahren erreicht wird.

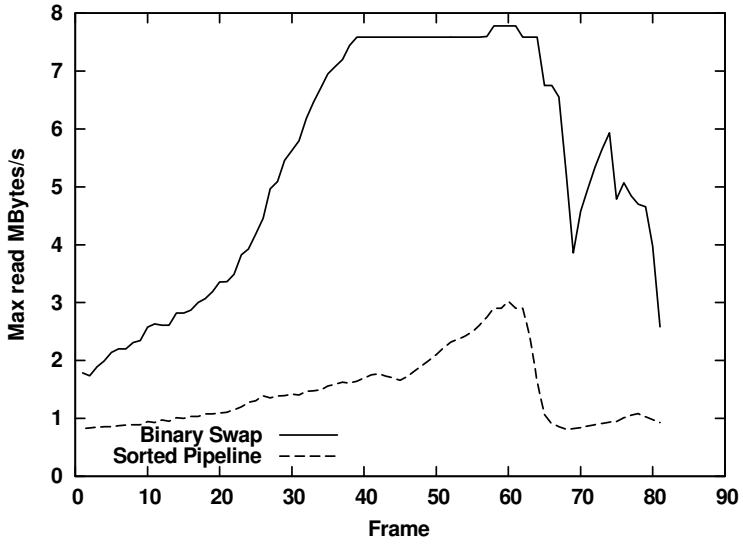

Abbildung 6-24: Maximal übertragene Datenmenge pro Knoten

Abbildung 6-24 zeigt für einen Flug durch das Power-Plant-Modell die maximale Datenmenge an, die von einem einzelnen Rechner übertragen wurde. I.d.R. bildet das Netzwerk den limitierenden Faktor bei der Bildkomposition. Aus diesem Grund wird die Geschwindigkeit der Bildkomposition maßgeblich von dem Rechner bestimmt, der die größte Datenmenge übertragen muss. Das Sorted-Pipeline-Verfahren erfordert für den gesamten Animationszyklus einen wesentlich geringeren maximalen Datenfluss. In Abbildung 6-29 ist die bei diesem Test verwendete Animation zu sehen. Beim Sorted-Pipeline-Verfahren muss insbesondere dann, wenn die Szene den Bildbereich vollständig überdeckt, nur ein Bruchteil der Datenmenge übertragen werden (wie beim Binary-Swap-Verfahren). Häufig wird in der Literatur die Effizienz einer Sort-Last-Bildkomposition mit komplexen Objekten demonstriert, die nur einen Teil der Bildebene überdecken. Durch die Datenreduktion, die mit einem Sort-Last-Sparse-Ansatz erreicht werden, können mit diesen Daten oft interaktive Bildwiederholraten erzielt werden. In realen Applikationen müssen jedoch häufig bildschirmfüllende Szenen dargestellt werden. Beispielsweise ist bei einer Innenansicht eines Architekturmodells immer der vollständige Bildbereich mit sichtbaren Objekten gefüllt. Allein mit Sort-Last-Sparse ist eine hochauflösende Darstellung in interaktive Bildwiederholraten nicht möglich. Das Sorted-Pipeline-Verfahren reduziert die transferierte Datenmenge so weit, dass auch in bildschirmfüllenden Darstellungen interaktive Bildwiederholraten möglich sind.

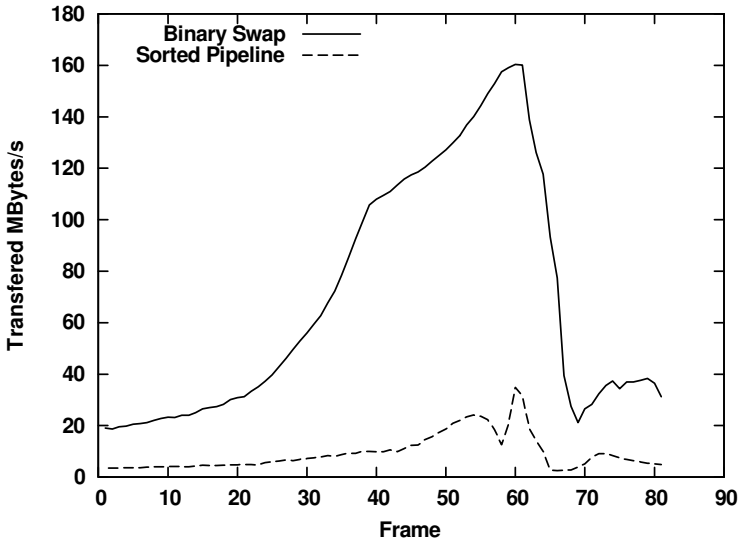

Abbildung 6-25: Übertragene Daten beim Flug durch das Power-Plant-Modell.

Bei der Bildkomposition über ein Standard-Ethernet-Netzwerk ist die maximale Datenrate durch die Geschwindigkeit limitiert, mit der ein Rechner mit dem Netzwerk verbunden ist. Abbildung 6-24 zeigt, in welchem Maße das Sorted-Pipeline-Verfahren die erforderliche Datenrate reduziert. In einem Netzwerk, das über einen zentralen Switch verbunden ist, besteht jedoch häufig noch ein weiterer limitierender Faktor. Die Kommunikation zwischen den Anschlüssen eines Switches erfolgt meist über einen gemeinsamen Datenbus. Die Geschwindigkeit des Datenbusses limitiert die summierte Datenrate aller angeschlossenen Rechner. Aus diesem Grund ist die Performanz nicht nur von der Datenrate der einzelnen Rechner abhängig, sondern auch von der aufsummierten Datenrate aller beteiligten Rechner. Abbildung 6-25 zeigt den Vergleich der summierten Datenrate für das Binary-Swap- und das Sorted-Pipeline-Verfahren. Beim Sorted-Pipeline-Verfahren müssen wesentlich weniger Daten übertragen werden. Dies ermöglicht den Einsatz eines günstigeren Switches oder den Aufbau größerer Cluster.

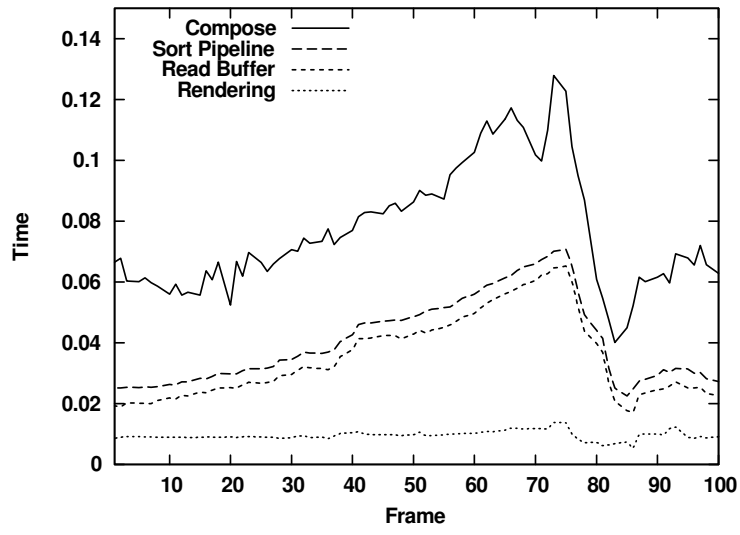

Abbildung 6-26: Power Plant mit Sorted-Pipeline-Komposition

Um eine Aussage über die Performanz des neuen Verfahrens zu treffen, wird für die oben beschriebenen Animationen untersucht, welchen Anteil an der Bildberechnung die einzelnen Arbeitsschritte haben.

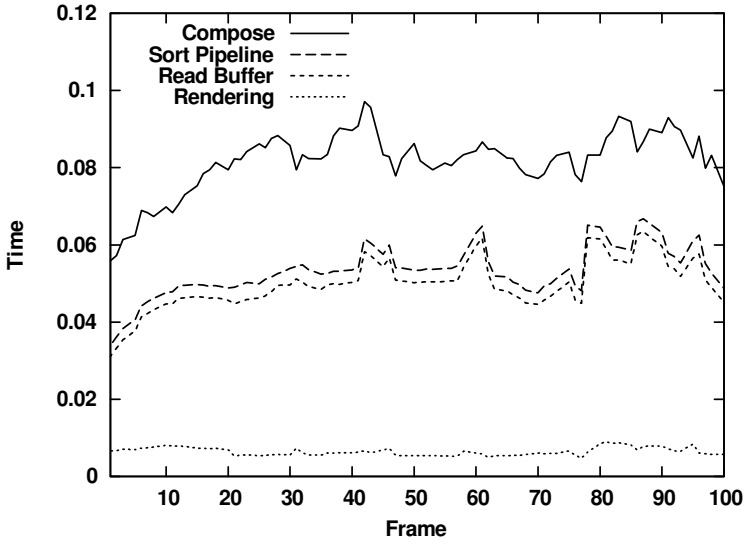

Abbildung 6-27: BMW mit Sorted-Pipeline-Komposition

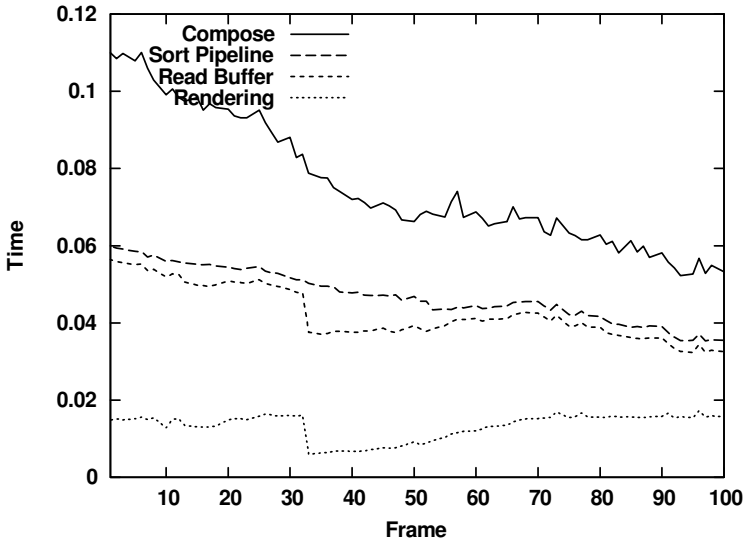

Abbildung 6-28: Lucy mit Sorted-Pipeline-Komposition

Abbildung 6-26 bis Abbildung 6-28 zeigen, dass die Sortierung der Pipelines und die Bildkomposition meist nur die Hälfte der Berechnungszeit ausmachen. Ein großer Teil der Verarbeitungszeit muss für das Auslesen und die Analyse des Bildspeichers aufgewendet werden. Dieser Verarbeitungsschritt ist stark von der eingesetzten Grafikkarte und von der Leistung des Prozessors abhängig und lässt sich nicht weiter optimieren.

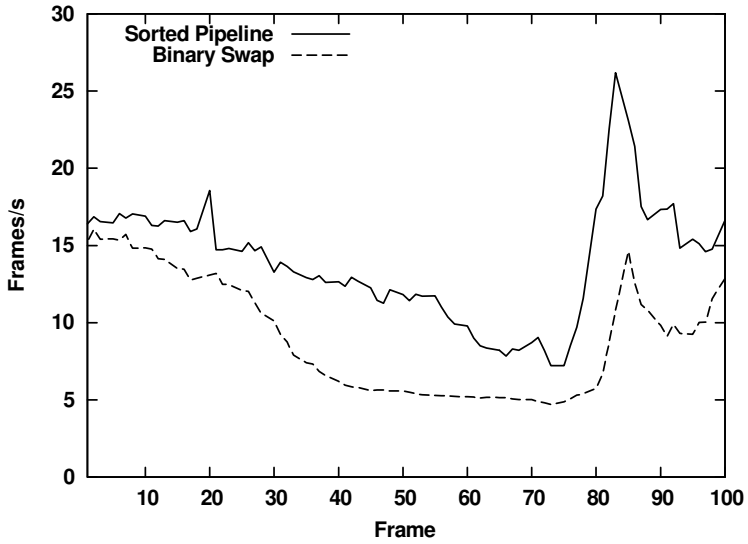

Abbildung 6-29: Power Plant. Vergleich Sorted-Pipeline mit Binary-Swap.

Abbildung 6-29 bis Abbildung 6-31 zeigen den direkten Vergleich der beiden Bildkompositionsverfahren. Es ist deutlich zu sehen, dass die Sorted-Pipeline-Bildkomposition wesentlich höhere Bildraten liefert. Die Steigerung der Performanz ist dann am größten, wenn die Szene große Teile des Bildes überdeckt. Im Extremfall muss bei der Binary-Swap-Bildkomposition von jedem Rechner der vollständige Bildbereich mit Z-Buffer ausgelesen und übertragen werden. Bei der Sorted-Pipeline-Bildkomposition kann bei der Darstellung bildschirmfüllender Szenen meist die Zahl der Pipelinestufen drastisch reduziert werden, wenn Objekte im Bildvordergrund Objekte im Hintergrund vollständig überdecken.

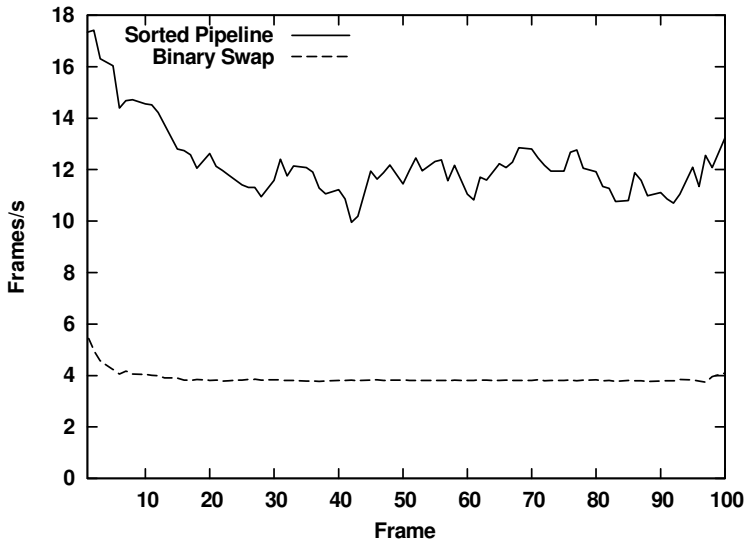

Abbildung 6-30: BMW7. Vergleich Sorted-Pipeline mit Binary Swap

Betrachtet man die in Abbildung 6-24 und Abbildung 6-25 erforderlichen Datenvolumen und vergleicht dies mit dem erzielten Performanz-Gewinn, so ist zu sehen, dass das Binary-Swap-

Verfahren trotz wesentlich höherer Datenübertragung nicht in gleichem Maße langsamer ist. Dies lässt sich anhand der Abbildung 6-10 erklären. Bei der Binary-Swap-Bildkomposition werden sehr große Datenbereiche über das Netzwerk an wenige Empfänger kopiert. Demgegenüber werden beim Sorted-Pipeline-Verfahren viele kleine Bildbereiche an sehr viele unterschiedliche Empfänger gesendet. Dieses Kommunikationsmuster ist für ein Ethernet-Netzwerk nicht optimal. Da jedoch einige leistungsfähige Netzwerk-Systeme für den Einsatz in Clustern mit hohen Bandbreiten und kurzer Latenz zur Verfügung stehen und in absehbarer Zeit auch preisgünstig eingesetzt werden können, ist zu erwarten, dass das Sorted-Pipeline-Verfahren mit einem effizienteren Netzwerk noch wesentlich höhere Bildwiederholraten mit hohen Bildschirmauflösungen erlauben wird.

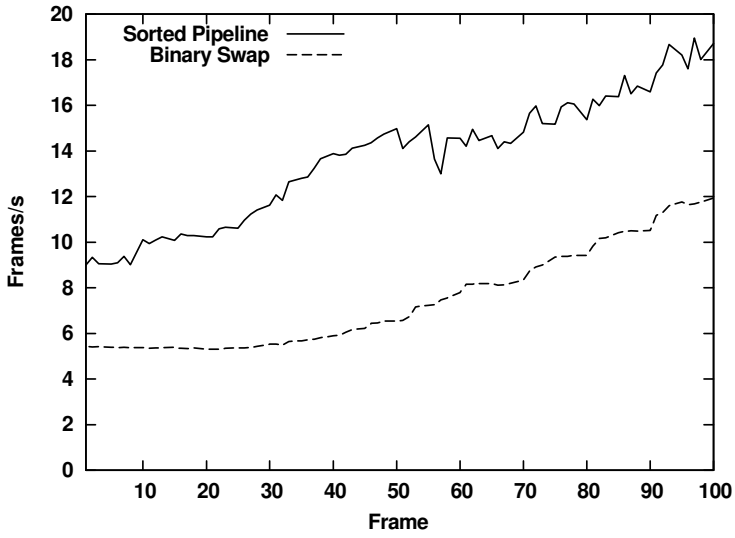

Abbildung 6-31: Lucy: 22 Mil. Polygone (Stanford Scanning Repository)

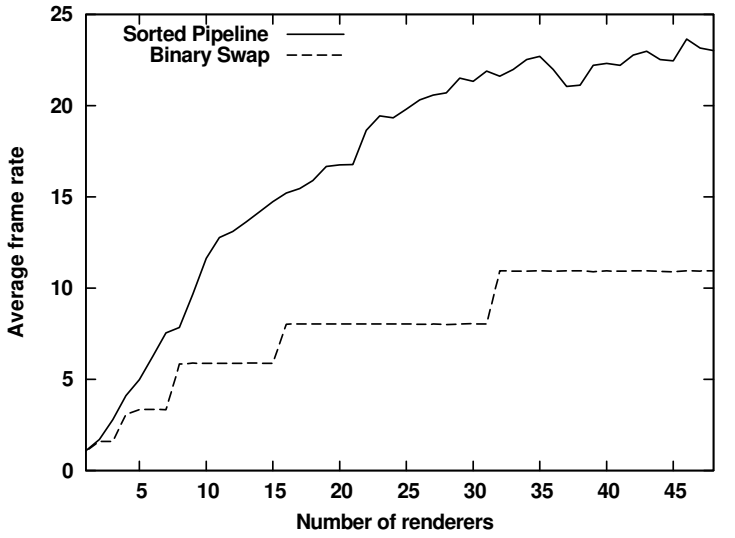

Abbildung 6-32: Lucy-Animation mit 1 - 48 PCs

Bisher wurde beim Vergleich der beiden Verfahren eine wesentliche Eigenschaft des Pipeline-Ansatzes nicht berücksichtigt. Eine Pipeline kann eine beliebige Länge besitzen, während das Binary-Swap-Verfahren eine Anzahl von n^2 Rechnern erfordert.

Dies bedeutet jedoch einen erheblichen Kostenfaktor. Für die Performanz-Tests in dieser Arbeit stand beispielsweise ein Cluster mit 48 Rechnern zur Verfügung. Um beide Verfahren vergleichen zu können, wurden nur 32 der Rechner eingesetzt. Mit dem Sorted-Pipeline-Verfahren können jedoch auch problemlos alle 48 Rechner zur parallelen Bildberechnung eingesetzt werden.

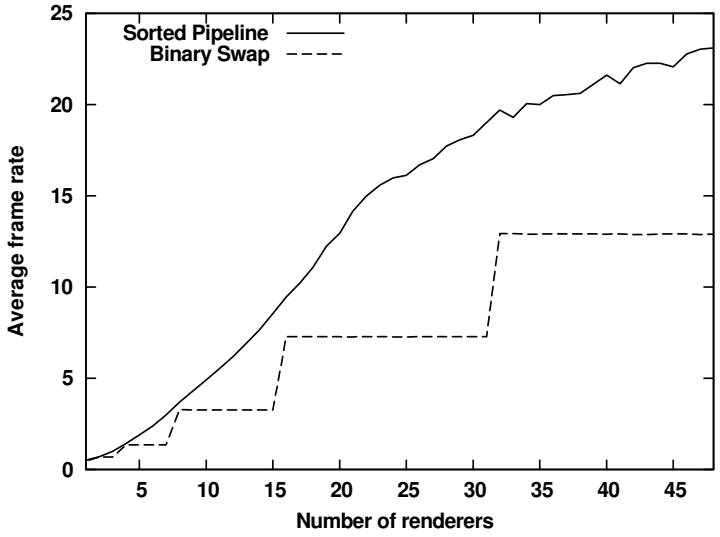

Abbildung 6-33: David: Bildwiederholrate bei 1 bis 48 Rechnern

6.4 Zusammenfassung

In diesem Kapitel wurden verschiedene Bildkompositionsverfahren auf ihre Eignung für Cluster-Systeme untersucht. Unter den in der Literatur zu findenden Kompositionsverfahren sind die Pipeline-Komposition und die Binary-Swap-Bildkomposition die beiden Verfahren, die die geringste Anforderung an das Netzwerk stellen und somit die beste Performanz in lose gekoppelten parallelen Systemen versprechen.

Auf der Basis der parallelen Kompositionspipeline wurde ein neues Bildkompositionsverfahren entwickelt, das Sorted-Pipeline-Verfahren, das in der Lage ist, den anfallenden Datenstrom wesentlich zu reduzieren. Ein Vergleich mit dem sehr effizienten und weit verbreiteten Binary-Swap-Verfahren hat gezeigt, dass es möglich ist, die Bildwiederholrate mit Hilfe der neu entwickelten Sorted-Pipeline-Komposition um den Faktor 2 bis 4 zu beschleunigen. Bei der Verwendung von Netzwerken mit geringerer Latenz ist eine weitere Beschleunigung gegenüber anderen Kompositionsverfahren zu erwarten.

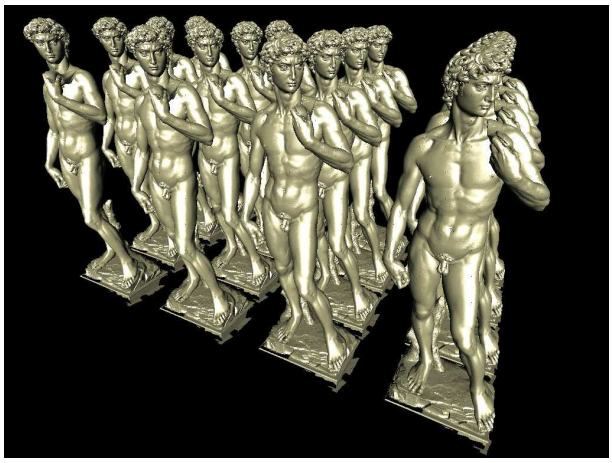

Abbildung 6-34: Darstellung einer Szene mit 900 Millionen Dreiecken

Bei der Sorted-Pipeline-Bildkomposition berechnet jeder Rechner in jedem Bild immer den gleichen Teil der Szene. Dadurch ist es möglich, sehr große Szenen auf viele Rechner zu verteilen. Hierbei muss kein Rechner die komplette Szene enthalten. Dies ist beispielsweise bei dem hybriden Sort-First- und Sort-Last-Verfahren oder bei der Layered Image Composition nicht möglich.

Das vorgestellte Verfahren ist allgemein für alle aus Polygonen zusammengesetzte Szenen geeignet. Das Verfahren liefert sehr gute Ergebnisse unabhängig von der Größe und Verteilung der Objekte in der Szene. Transparente Objekte werden korrekt dargestellt. Der darzustellende Szenengraph muss nicht verändert werden.

7 Anwendung

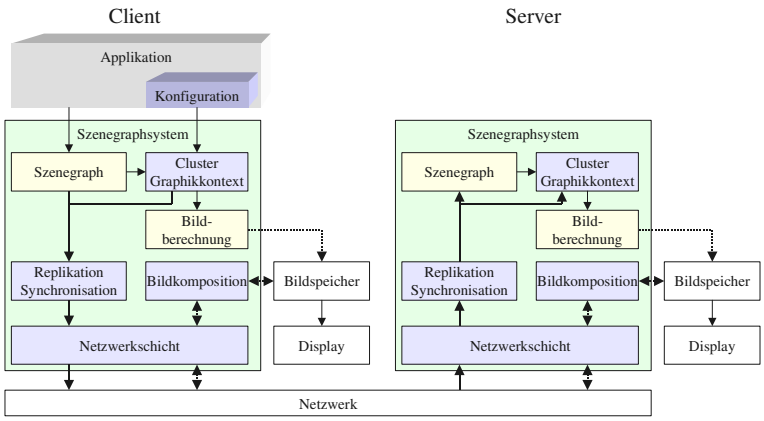

Abbildung 7-1: Anwendung und Konfiguration des Cluster-Szenengraphen

Bisher wurde beschrieben, wie eine Szenenbeschreibung in einem Cluster verteilt und parallel berechnet werden kann. Alle hierfür erforderlichen Verfahren wurden in das Szenengraphensystem OpenSG integriert. Dieses Kapitel beschreibt nun den Einsatz des Szenengraphen für unterschiedliche Anwendungsszenarien. Es wird gezeigt, wie beliebige Projektionssysteme mit den in dieser Arbeit entwickelten Techniken angesteuert und sehr große Modelle parallel geladen und berechnet werden können. Am Beispiel des VR-Systems Avalon wird die Integration des Cluster-Szenengraphen in ein VR-System aufgezeigt. Zusätzlich wird eine komfortable interaktive Konfigurationsschnittstelle beschrieben.

7.1 Projektionssysteme mit mehreren Einzelprojektionen

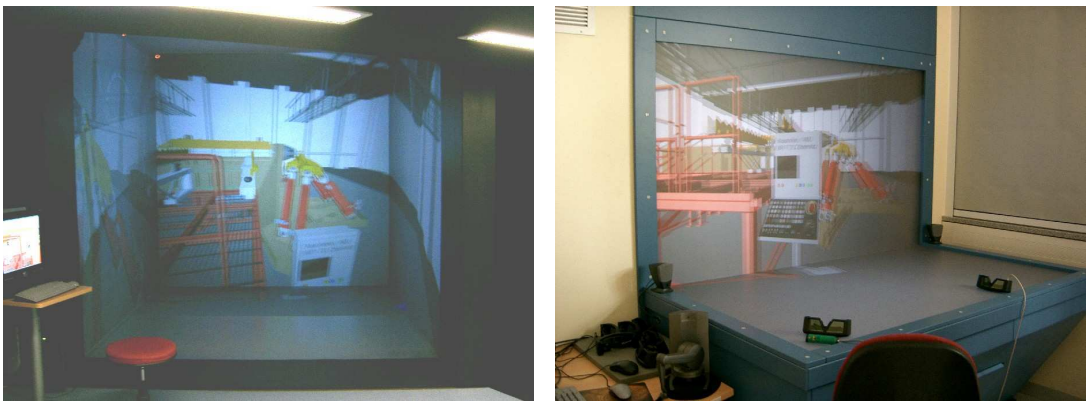

Abbildung 7-2: Cave und Workbench im IPK Chemnitz

Cluster-Systeme werden häufig eingesetzt, um Projektionssysteme mit mehreren Projektoren anzusteuern. Für die Bildberechnung in einer Cave oder auf einem zusammengesetzten Display muss die gleiche Szene aus unterschiedlichen Blickrichtungen auf eine Leinwand projiziert werden. Alle Leinwände zusammen ergeben für einen Betrachter ein zusammenhängendes Bild der Szene. Solche Projektionssysteme lassen sich sehr gut mit einem Cluster realisieren. Hierbei wird jeder

Projektor von einem Rechner des Clusters angesteuert. Jeder Rechner berechnet nur den Ausschnitt, der auf der jeweiligen Leinwand zu sehen ist. Die Bildberechnung in einer solchen Konfiguration entspricht der in Kapitel 5 beschriebenen Sort-First-parallelen Bildberechnung mit einer statischen Bildunterteilung. In Abbildung 7-2 und Abbildung 7-3 sind Beispiele für Projektionssysteme mit mehreren Projektoren zu sehen.

Projektionssysteme können mit Hilfe beliebig vieler Projektoren betrieben werden. Der Einsatz mehrerer Projektoren ermöglicht es, den Grad der Immersion zu erhöhen. Einerseits kann der Blickwinkel, in dem die Szene betrachtet werden kann, erweitert werden, oder eine Szene kann mit Hilfe mehrerer Projektoren in einer erhöhten Auflösung dargestellt werden. Cave und gekachelte Projektionen sind die bekanntesten Vertreter der Multi-Projektor-Projektionssysteme. Es lassen sich jedoch auch Rund-, Kuppel- und beliebige andere Projektionssysteme nach dem gleichen Prinzip aufbauen.

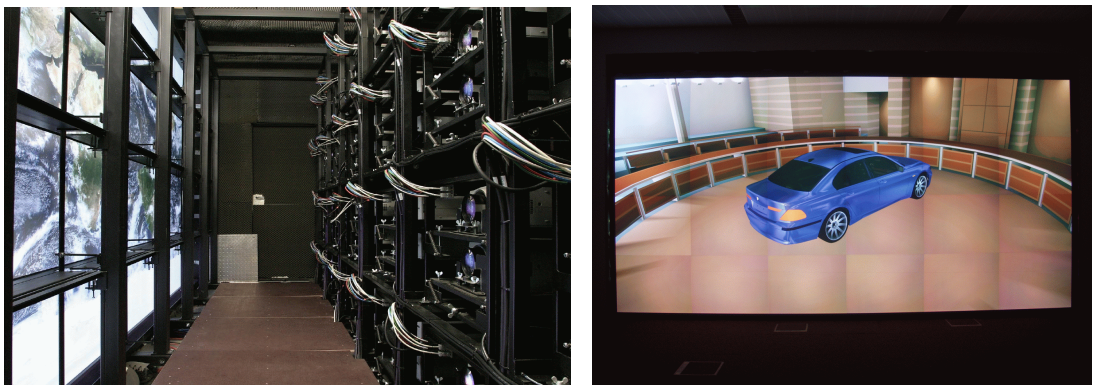

Abbildung 7-3: HEyeWall mit 48 Projektoren im Fraunhofer IGD Darmstadt

Aufgrund der großen Vielfalt muss eine Ansteuerung von Multi-Projektor-Projektionssystemen sehr flexibel gestaltet sein. Im Folgenden wird beschrieben, wie mit den Techniken, die im Rahmen dieser Arbeit entwickelt wurden, jedes Projektionssystem angesteuert werden kann. Am Beispiel einer einfachen gekachelten Projektion wird das Vorgehen erläutert. Anschließend wird der Einsatz für beliebige Projektionssysteme beschrieben.

7.1.1 Gekachelte Projektionen

Mit Hilfe eines Szenengraphensystems lassen sich verschiedene Ansichten einer oder mehrerer Szenen in einem einzelnen Fenster darstellen. In Applikationen zum Editieren von Szenen wird diese Funktion meist eingesetzt, um eine Szene gleichzeitig aus verschiedenen Richtungen darzustellen. Die Konfiguration solcher Ansichten in einem Fenster erfolgt mit Hilfe so genannter Viewports. Ein Viewport ist ein rechteckiger Ausschnitt im Darstellungsbereich. Dieser Ansatz dient als Grundlage für die Konfiguration eines Projektionssystem mit mehreren Projektoren. Position, Lage und Eigenschaften eines Fensters zur Darstellung einer Szene werden in einem Szenengraphensystem mit Hilfe einer Datenstruktur beschrieben. In OpenSG wird diese Datenstruktur als *Window* bezeichnet. Ein *Window* kann einen oder mehrere Viewports beinhalten. Für den Einsatz in einem Cluster wird als Erweiterung die neue Datenstruktur *ClusterWindow*

eingeführt. Diese Datenstruktur unterscheidet sich von einem normalen *Window* dadurch, dass die Visualisierung auf beliebigen Rechnern in einem Cluster durchgeführt werden kann.

In einem *ClusterWindow* überprüft jeder Rechner für jeden Viewport, ob eine Überlappung mit dem Zuständigkeitsbereich vorliegt. Ist dies nicht der Fall, kann der entsprechende Viewport ignoriert werden. Existiert hingegen eine Überlappung, so muss eine Bildberechnung für die Schnittmenge aus Viewport und Zuständigkeitsbereich erfolgen. Abbildung 7-4 zeigt dies am Beispiel einer gekachelten 3x3-Projektion.

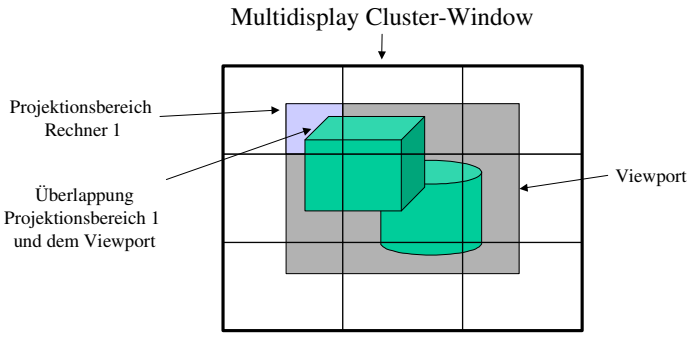

Abbildung 7-4: Multi-Projektor *ClusterWindow*

Um einen Ausschnitt eines Viewports darzustellen, muss die Projektionsmatrix des aktuellen Betrachtungspunktes so modifiziert werden, dass die ursprüngliche Projektion zwar erhalten bleibt, jedoch nur der Überlappungsbereich sichtbar ist. Die Modifikation der Matrix erfolgt mit Hilfe so genannter *CameraDecorator*s. Ein *CameraDecorator* nimmt eine bestehende Projektionsmatrix und fügt ihr bestimmte Eigenschaften hinzu. Um einen Bildausschnitt darzustellen, wird beispielsweise ein *TileDecorator* verwendet.

$$\begin{pmatrix} \frac{1}{right-left} & 0 & 0 & -\frac{left*2-1}{right-left}-1 \\ 0 & \frac{1}{top-bottom} & 0 & -\frac{bottom*2-1}{top-bottom}-1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Formel 1: Matrix des TileDecorators

Jeder Rechner, auf dem ein Teil eines Viewports zu sehen ist, fügt der aktuellen Kamera einen *Decorator* hinzu, so dass nur der Bereich sichtbar ist, der mit dem Darstellungsbereich eines einzelnen Rechners überlappt. Fügt man alle projizierten Bilder zusammen, so ergibt sich wieder ein vollständiges Bild. Mit Hilfe des *ClusterWindows* lassen sich gekachelte Projektionen sehr einfach realisieren. Die Ansteuerung ist für die Applikation identisch mit einer Ansteuerung eines einzelnen lokalen Darstellungsfensters. Die Projektionsbereiche der Rechner des Clusters werden in einem *ClusterWindow* zusammengefasst und aus Sicht der Applikation als einheitliche Projektionsebene betrachtet.

Auf den ersten Blick ist das Multidisplay *ClusterWindow* nur für gekachelte Mono-Projektionen einsetzbar. Im Folgenden wird gezeigt, dass das bisher vorgestellte Konzept auch für die Ansteuerung beliebiger Projektionssysteme eingesetzt werden kann.

7.1.2 Ansteuerung beliebiger Projektionssysteme

Gekachelte Projektionen sind nur eines von vielen Projektionssystemen, die mit mehreren Einzelprojektionen aufgebaut werden können. Mehrere Projektionen werden verwendet, wenn ein einzelner Projektor die geforderte Auflösung nicht erreichen kann oder wenn die Projektionsfläche nicht eben ist. Da einzelne Projektionen beliebig zusammengefügt werden können, ist es praktisch nicht möglich, für jedes mögliche Projektionssystem eine vorgefertigte Konfiguration anzubieten. Allein bei der Ansteuerung einer Cave müssen beispielsweise folgende Parameter berücksichtigt werden:

- Anzahl der Projektionswände
- Abmessungen der Cave
- Auflösung der Projektoren
- Anordnung der Projektoren
- Ausrichtung der Projektoren
- Aktiv- oder Passiv-Stereo
- Größe des Projektionsbereichs
- Versatz zwischen rechtem und linkem Auge

Selbst wenn es bei der Ansteuerung einer Cave noch möglich ist, mit einer überschaubaren Anzahl an Parametern eine automatische Konfiguration vorzunehmen, besteht die Gefahr, dass durch den Automatismus mögliche Konfigurationen ausgeschlossen werden. Es wäre beispielsweise denkbar, jede Wand einer Cave mit Hilfe von vier Projektoren aufzubauen, um eine höhere Auflösung zu erreichen.

Um bei der Konfiguration alle möglichen Konstellationen zu unterstützen, wird in dem hier vorgestellten Ansatz auf eine Spezialisierung des *ClusterWindow* zu Gunsten eines allgemeineren Ansatzes verzichtet. Im Folgenden wird gezeigt, wie die oben vorgestellten Mechanismen auf einfache Weise dazu verwendet werden können, beliebige Projektionssysteme zu betreiben. Hierzu werden die folgenden zusätzlichen *CameraDecorator*s benötigt.

- *TileDecorator*: Aus einer gegebenen Projektionstransformation wird ein rechteckiger Bereich ausgewählt.
- *ProjektionDecorator*: Verändert die Projektionsmatrix für eine Projektion aus der Sicht eines Betrachters. Dieser *Decorator* wird für eine Cave oder ähnliche Projektionen benötigt, bei denen die Darstellung von der Position des Betrachters abhängig ist.

- *StereoDecorator*: Verändert die Projektionsmatrix so, dass entweder ein Bild für das linke oder das rechte Auge berechnet wird.
- *MatrixDecorator*: Dieser *Decorator* erlaubt beliebige Manipulationen der Projektionsmatrix wie z.B. Skalierung oder Scherung.

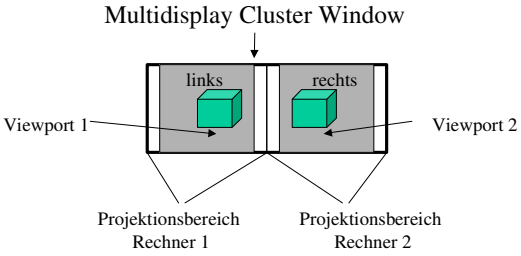

Abbildung 7-5: Stereoprojektion mit zwei Rechnern

In Projektionssystemen, die nicht aus gekachelten Projektionen bestehen, muss die Kamera-Transformation jeder Projektion individuell gesetzt werden. In einer Cave muss sie beispielsweise in Abhängigkeit von der Position und Große der Leinwände und von der Position des Betrachters für jede Wand unterschiedlich gesetzt werden. Um jeden Rechner, der an der Bildberechnung beteiligt ist, mit einer speziellen Projektionsmatrix zu versehen, wird in das *ClusterWindow* für jeden Rechner ein Viewport eingefügt. Dieser Viewport liegt vollständig im Zuständigkeitsbereich der Einzelprojektion. Jeder Viewport wird dadurch genau auf einem Rechner berechnet und dargestellt. Jedem Viewport wird anschließend einer oder mehrere *Decorators* zugewiesen. Abbildung 7-5 und Abbildung 7-6 zeigen dies für eine Stereo-Projektion und eine 5-Seiten Cave.

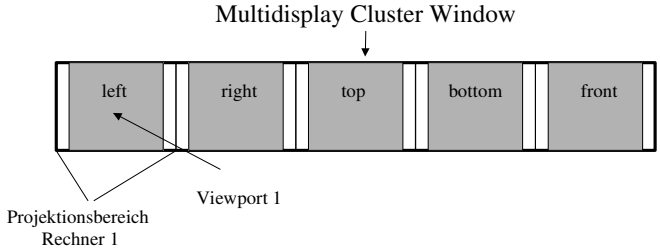

Abbildung 7-6: Konfiguration einer 5-Seiten Cave

Diese Technik wird häufig auch auf einem einzelnen Rechner eingesetzt, wenn mit Hilfe einer Grafikkarte mit zwei Ausgängen eine passive Stereoprojektion angesteuert werden soll. Besitzen die Projektionen für das rechte und das linke Auge jeweils eine Auflösung von 1024 x 768 Bildpunkten, dann wird die Größe des Grafikkartenspeichers auf eine Auflösung von 2048 x 768 Bildpunkten eingestellt und die beiden Ausgänge so konfiguriert, dass jede Projektion eine Hälfte des virtuellen Bildschirms darstellt. In dem Fenster, das den gesamten Bildschirm überdeckt, werden zwei Viewports eingefügt. Der Viewport für das rechte Auge überdeckt den Bereich von 0,0 bis 1023,768, und der Viewport für das linke Auge überdeckt den Bereich von 1024,0 bis 2047,768. In beiden Viewports wird die gleiche Szene mit unterschiedlichen Kameraparametern

dargestellt. Da diese Technik weit verbreitet ist, ist es naheliegend, bei der Konfiguration von Projektionen im Cluster ähnlich vorzugehen. Die Größe des *ClusterWindows* wird so eingestellt, dass alle Projektionen nebeneinander angeordnet werden können. In einer passiv angesteuerten 5-Seiten-Cave werden beispielsweise zehn Projektionen benötigt. Besitzt jede Projektion eine Auflösung von 1280 x 1024 Bildpunkten, so wird die Größe des *ClusterWindows* auf 12800 x 1024 Bildpunkte eingestellt. Anschließend werden in das Fenster zehn Viewports nebeneinander eingefügt. Im Falle der Stereoprojektion mit einer Grafikkarte muss der Treiber so konfiguriert werden, dass an jedem Ausgang nur ein Teil des gesamten Bildes sichtbar ist. Im Falle des *ClusterWindows* wird dies per Software gesteuert.

Wie oben beschrieben, wird durch das *ClusterWindow* von jedem Rechner ein Teil des Bildes berechnet. Liegen alle Viewports nebeneinander und gibt es genau so viele Viewports wie Rechner für die Bildberechnung, dann übernimmt jeder Rechner exakt einen Viewport. Im Gegensatz zur Stereoprojektion mit einer Grafikkarte werden vom *ClusterWindow* alle Viewports parallel berechnet. Die Performanz der Bildberechnung ist daher unabhängig von der Anzahl der Projektionen. Dadurch können auch sehr komplexe Projektionen effizient betrieben werden. Beispielsweise besteht die HEyeWall am Fraunhofer IGD in Darmstadt aus 48 Einzelprojektionen. Mit den im Rahmen dieser Arbeit entwickelten Mechanismen ist es möglich, auf diesem Projektionssystem Bildwiederholraten von über 100 Bildern/s zu erreichen.

7.1.3 Farbkorrektur

Werden für die Darstellung einer Szene mehrere Projektoren verwendet, so ergibt sich für einen Betrachter nur dann ein zusammenhängendes homogenes Bild, wenn Helligkeit und Farbverteilung aller Projektoren identisch sind. Das menschliche Auge ist in der Lage, bereits sehr geringe Farbunterschiede festzustellen. Unterschiede in Helligkeit oder Farbe der einzelnen Projektionen führen dazu, dass die Ränder der Projektion deutlich wahrgenommen werden.

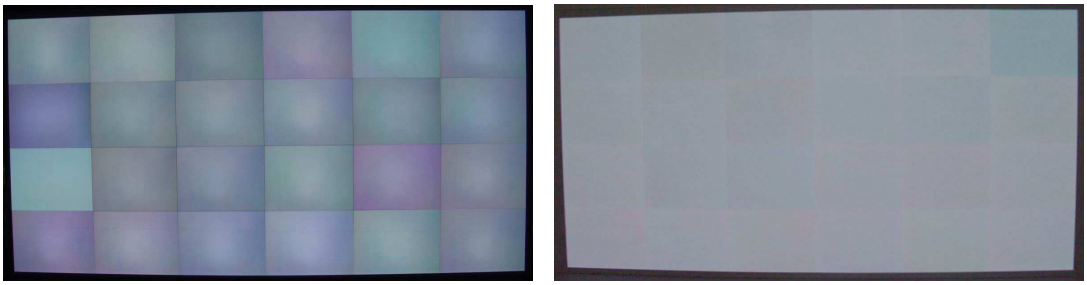

Abbildung 7-7: Tiled Display ohne und mit Farbkorrektur

Um dieses Problem zu beheben, werden die berechneten Bilder mit Hilfe einer Gammatabelle und einer Farbtransformationsmatrix korrigiert. Die Gammatabelle dient dazu, die Helligkeitswerte des Bildspeichers und die Helligkeit der Projektion linear aneinander anzugleichen. D.h., eine Verdoppelung des Helligkeitswertes im Bildspeicher führt zu einer doppelt so hellen Projektion. Anschließend werden die Farbwerte (rot, grün, blau) mit Hilfe einer Korrekturmatrix transformiert. Für jede Projektion werden Gammatabelle und Farbmatrix durch eine Messung bestimmt. Jeder

Rechner, an dem eine Projektion angeschlossen ist, führt eine Farbkorrektur des Bildspeichers durch.

Die im Rahmen dieser Arbeit entwickelten Techniken ermöglichen eine exakte Anpassung der Farbräume unterschiedlicher Projektoren. Die Ermittlung der Parameter für die Farbkorrektur ist jedoch nicht Teil dieser Arbeit. Hierfür wird auf die Arbeiten von Majumder [MaS04] und Kresse [KRK03] verwiesen. In diesen Arbeiten wird beschrieben, wie die Projektionsfläche mit Hilfe eines Farbmessgerätes vermessen und daraus Parameter für die Farbkorrektur gewonnen werden können.

7.1.4 Geometrische Anpassung der Projektion

Heute am Markt verfügbare Projektoren bieten nur sehr geringe Möglichkeiten, das projizierte Bild zu verzerren. Dies ist erforderlich, um z.B. auf gekrümmte Flächen zu projizieren oder um Fehler in der Optik eines Projektors auszugleichen. Einige der früher sehr verbreiteten Röhrenprojektoren ließen sich sehr gut nachträglich justieren. Allerdings sind Röhrenprojektoren lichtschwach, sehr teuer, und mittlerweile existiert mit der Firma Barco nur noch ein einziger Hersteller für diese Projektoren. Alle anderen am Markt erhältlichen Projektoren sind DLP- oder LCD-Projektoren. Das projizierte Bild diese Projektoren ist meist sehr hell, sie lassen sich jedoch nur sehr eingeschränkt justieren.

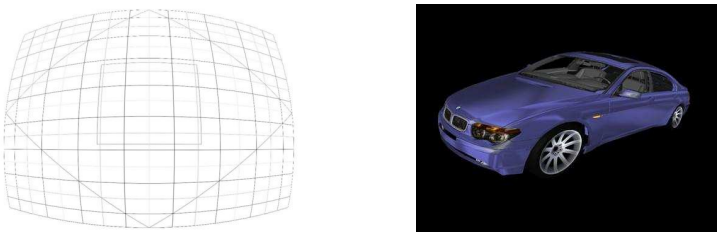

Abbildung 7-8: Geometrische Anpassung der Projektion

Um dieses Problem zu lösen, wird der Bildberechnung ein weiterer Schritt hinzugefügt. Das Bild wird wie bisher berechnet und in den Bildspeicher geschrieben. Es wird jedoch nicht mehr direkt angezeigt, sondern in eine Textur auf der Grafikkarte kopiert und anschließend auf ein Gitter projiziert. Die Knotenpunkte dieses Gitters können beliebig versetzt werden. Damit lassen sich Fehler in der Optik eines Projektors sehr gut ausgleichen. Das Bild kann beliebig auf der Projektionsfläche verschoben, global oder lokal gestreckt, gestaucht oder rotiert werden.

Mit Hilfe dieser geometrischen Korrektur des Bildes ist es möglich, sehr günstige und lichtstarke Projektoren einzusetzen. Das projizierte Bild kann auch ohne die Unterstützung des Projektors sehr genau auf der Leinwand ausgerichtet werden. Die Möglichkeit der geometrischen Anpassung kann auch als Basis für eine automatische Kalibrierung eingesetzt werden, wie sie beispielsweise in [CSW02] vorgeschlagen wird.

7.1.5 Cave-Projektion

Als erstes komplexeres Projektionssystem soll die Cave des Fraunhofer-IGD in Darmstadt mit den in dieser Arbeit entwickelten Techniken angesteuert werden. Die Cave wird von zehn PCs betrieben, an denen zehn DLP-Projektoren mit einer Auflösung von jeweils 1600 x 1200 Bildpunkten angeschlossen sind. Die Trennung der Bilddaten für das linke und rechte Auge erfolgt mit Infitec-Filtern. Aufgrund der beengten Platzverhältnisse erfolgt die Projektion mit einer Weitwinkellinse über einen Spiegel.

Der Projektionsbereich der Projektoren ist größer als die Leinwand, auf der das Bild dargestellt werden soll. Aus diesem Grund muss für jede Wand der Cave der sichtbare Bildbereich separat konfiguriert werden. Die Basis für die Konfiguration bildet das *ClusterWindow*. Für jeden Projektor wird ein rechteckiger Projektionsbereich definiert. Größe und Position können sehr genau an die Leinwand angepasst werden.

Aufgrund des eingesetzten Linsensystems und der Spiegel ist das projizierte Bild an den Rändern verzerrt. Da mit Hilfe der eingesetzten Projektoren keine Entzerrung der Projektion möglich ist, muss dies mit Hilfe der oben beschriebenen geometrischen Entzerrung per Software durchgeführt werden. Auf jede Projektionswand wird ein Ausgleichsgitter projiziert. Dieses Gitter wird manuell so ausgerichtet, dass die Projektion ohne Verzerrungen die Leinwand überdeckt. Durch die Stauchung und Verzerrung einzelner Bildteile können durch die Entzerrung sichtbare Artefakte entstehen. Diese sind für einen Betrachter jedoch weniger störend als eine verzerrte Projektion.

Die Verwendung von Infitec-Farbfiltern zur stereoskopische Darstellung bewirkt eine sichtbare Farbabweichung zwischen den Darstellungen für das rechte und linke Auge. Dies lässt sich durch eine Anpassung der Farbwerte weitgehend beheben. Hierzu wird jeder Projektor photometrisch vermessen und alle mit Hilfe einer Gammatabelle und einer Farbmatrix aneinander angeglichen. In jedem Rechner, der eine Projektion betreibt, werden die Farbwerte vor der Ausgabe entsprechend korrigiert.

Trotz einiger Besonderheiten lässt sich die Cave am Fraunhofer IGD in Darmstadt ohne Änderungen an den oben beschriebenen Konzepten problemlos konfigurieren und betreiben. Wird die Darstellung nicht an die Bildwiederholrate der Projektoren gekoppelt, lassen sich bei ausgeschalteter Farbkorrektur, trotz der erforderlichen Synchronisation über das Netzwerk, Bildwiederholraten von über 300 Bilder/s erreichen. Dies liegt weit über den Anforderungen interaktiver Echtzeitanwendungen. Mit eingeschalteter Farbkorrektur sinkt die Bildwiederholrate auf etwa 100 Bilder/s. Die relativ langsame Farbkorrektur ist auf die mittlerweile etwas betagten NVidia-Ti-4800-Grafikkarten zurückzuführen. Auf modernen Grafikkarten ist eine wesentlich schnellere Farbkorrektur möglich.

Im Vergleich zur früheren Cave im Fraunhofer IGD, die mit einer SGI-Infinite-Reality-Maschine mit drei Grafikpipelines betrieben wurde, ist die neue Cluster-basierte Version wesentlich

schneller, preisgünstiger und erlaubt die Darstellung komplexerer Modelle mit höheren Bildwiederholraten.

7.1.6 HEyeWall

Die HEyeWall ist ein hochauflösendes, stereoskopisches Projektionssystem, das mit dem Ziel entwickelt wurde, eine Display-Auflösung zu erreichen, die über der des menschlichen Auges liegt. Hierfür werden 48 Projektoren eingesetzt, die paarweise in sechs Spalten und vier Reihen angeordnet sind. Ein etwas kleineres System aus 30 Projektoren wurde auf der CeBIT 2004 der breiten Öffentlichkeit vorgestellt. Schon zu Beginn der Planungen stand fest, dass ein solches Projektionssystem nur dann erfolgreich vermarktet werden kann, wenn es aus preiswerten Standardkomponenten besteht. Aufgrund der guten Erfahrungen mit dem Betrieb der 5-Seiten-Cave sollte die HEyeWall mit einem Cluster aus 48 PCs betrieben werden.

Eine HEyeWall ist modular aufgebaut. Jedes Modul enthält zwei Projektoren mit Infitec-Filtern für die stereoskopische Projektion einer Kachel. Die Projektionsfläche wird durch Aneinanderreihung und Stapelung beliebig vieler Module gebildet. In zusammengesetzten Projektionen können die Bilder benachbarter Projektoren meist nicht auf den Bildpunkt genau justiert werden. Dies liegt einerseits daran, dass sich die Aufhängung der Projektoren durch thermische Ausdehnung verändert und andererseits daran, dass die Kanten einer Projektion meist keine exakte Linie darstellen. Aus diesem Grund wird die Projektion meist überlappt und mit Hilfe einer Elektronik die Helligkeit des Überlappungsbereichs heruntergeregelt. Für die HEyeWall ist aus Kostengründen eine elektronische Lösung nicht möglich. Als Alternative wird mit Hilfe von Blenden der Überlappungsbereich so geteilt, dass exakt immer nur ein Projektor eine Stelle der Leinwand beleuchtet. Hierzu werden alle Projektionen so eingestellt, dass sie sich um etwa vier Bildpunkte mit ihren Nachbarprojektionen überlappen

Die Konfiguration erfolgt wie bei der vorher beschriebenen Cave wieder mit Hilfe des *ClusterWindows*. Für jeden Projektor wird unter Berücksichtigung der überlappenden Projektion ein Projektionsbereich definiert. Bei der Installation der HEyeWall wird für die Zuweisung der Projektionsbereiche ein festes Raster verwendet. Es ist jedoch auch möglich, jeden Projektor einzeln zu definieren und somit Fehler in der Justierung auszugleichen.

Die Anwendung der in dieser Arbeit entwickelten Techniken in der Cave des Fraunhofer IGD und auf der HEyeWall zeigt, dass hochwertige Projektionen auch mit preisgünstigen Komponenten aufgebaut und betrieben werden können.

7.2 Darstellung großer Szenen

Dieser Abschnitt beschäftigt sich mit dem Problem der Darstellung sehr großer Szenen. Hierbei steht nicht die Parallelisierung der Bildberechnung im Vordergrund, sondern das Speichermanagement bei der Darstellung von Szenen, die nicht vollständig in den Hauptspeicher

eines einzelnen Rechners geladen werden können. Für die Darstellung dieser Szenen wird die in Kapitel 6 beschriebene Sort-Last-parallele Bildberechnung verwendet.

Große Szenen entstehen dann, wenn sehr viele Details sichtbar gemacht werden müssen oder wenn klassische Geometrieoptimierungen, wie sie in [Fus93] von Thomas Funkhauser u.a. beschrieben werden, nicht angewandt werden können. Da ein Display meist nur einige Millionen Bildpunkte besitzt, lassen sich Szenen aus mehreren hundert Millionen Polygonen meist ohne einen Verlust in der visuellen Qualität so stark reduzieren. Diese reduzierten Modelle können oft wieder vollständig in den Hauptspeicher eines Rechners geladen werden. Die Reduktion ist jedoch zeitaufwendig und häufig nicht ohne einen manuellen Eingriff möglich. Es ist daher in vielen Fällen sehr hilfreich, wenn große Szenen ohne weitere Vorarbeiten visualisiert werden können. Der Einsatz sehr hochauflösender Displays erschwert ebenfalls die Verwendung von Reduktionsverfahren, da durch die hohe Auflösung sehr viele Details sichtbar sind.

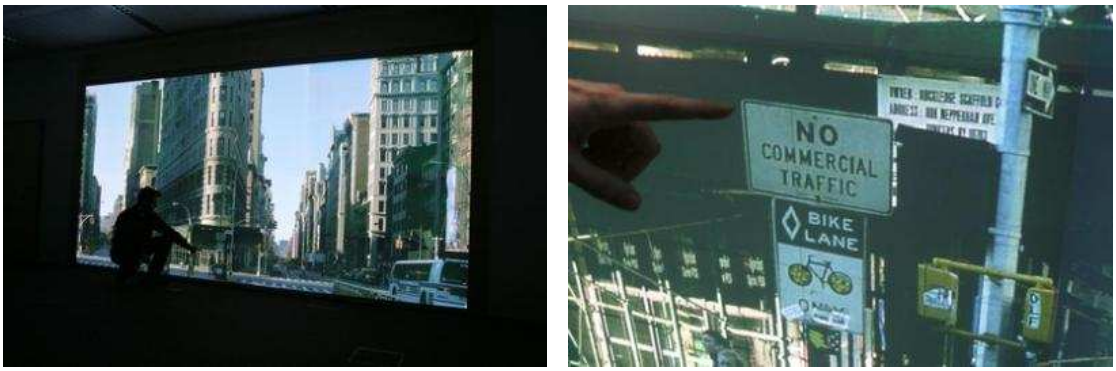

Abbildung 7-9: HEyeWall: Hochauflösendes Display am IGD-Darmstadt

Abbildung 7-9 zeigt auf der rechten Seite eine Nahaufnahme der hochauflösenden Projektionswand. Auf einem solchen Projektionssystem sind sehr viele Details sichtbar. Eine Vereinfachung der Modelle führt in vielen Fällen zu einem wahrnehmbaren Qualitätsverlust. In den Visionen von Thomas Funkhauser und Kai Li [FuL00] werden in Zukunft ganze Wände mit hochauflösenden Displays ausgefüllt. Um solche Projektionen sinnvoll zu nutzen, sind enorme Datenmengen zu bewältigen.

Es muss daher eine Lösung gefunden werden, um Szenen darzustellen, die nicht vollständig in den Hauptspeicher eines einzelnen Rechners geladen werden können. In der Literatur findet sich hierzu ein Lösungsansatz, bei dem die Geometrie erst bei Bedarf in den Hauptspeicher geladen wird. Bei knappem Speicherplatz werden nicht benötigte Teile der Szene ausgelagert. Diese Technik wird meist als Out-of-Core-Rendering bezeichnet [CKS02]. Mit dieser Technik ist die Größe einer Szene nur durch den zur Verfügung stehenden Plattenplatz beschränkt. Allerdings führt der relativ aufwendige Zugriff auf den externen Speicher zu einem langsameren Bildaufbau.

In einem Cluster-System werden viele Rechner mit lokalem Hauptspeicher eingesetzt. Verwendet man ausreichend viele Rechner, so steht in Summe häufig genügend Arbeitsspeicher zur

Verfügung. D.h., ein Modell kann, zerlegt in kleinere Teile, vollständig geladen werden. Dabei stellt sich das Problem, wie ein Szenengraph, der, wie in Kapitel 4 beschrieben, per Multicast an alle Rechner in einem Cluster verteilt wird, so modifiziert werden kann, dass einzelne Teile des Graphen nur von jeweils einem einzelnen Rechner in den Hauptspeicher geladen werden.

7.2.1 Stellvertretergruppen im Szenengraphen

Im Rahmen dieser Arbeit wurde ein Konzept umgesetzt, mit dessen Hilfe in einem Szenengraph Teilgraphen zu einem späteren Zeitpunkt nachgeladen werden können. Hierzu werden Stellvertretergruppen verwendet. Diese werden im Folgenden auch als *ProxyGroups* bezeichnet. Eine *ProxyGroup* ist ein Gruppenknoten des Szenengraphen. Er enthält einen Verweis auf eine Teilszene, die geladen und als Kind der Gruppe in den Graphen eingefügt werden soll.

Damit eine *ProxyGroup* vom Szenengraphensystem auch dann bearbeitet werden kann, wenn ein Teilbaum noch nicht geladen ist, werden neben dem Verweis auf die zu ladende Szene statistische Informationen gespeichert. Diese sind für den Sichtbarkeitstest und die Lastverteilung bei der parallelen Bildberechnung erforderlich. Die folgende Informationen werden gespeichert:

- Verweis auf die zu ladende Teilszene
- Größe und Position der Teilszene (Hüllvolumen)
- Anzahl der Eckpunkte
- Anzahl der Polygone
- Anzahl der geometrischen Objekte

Bei der Abarbeitung der *ProxyGroup* prüft das Szenengraphensystem, ob die zu ladende Teilszene im sichtbaren Bereich liegt. Ist dies der Fall, dann wird in einem eigenen Prozess der Ladevorgang begonnen. Ist die Teilszene nicht sichtbar oder ist der Gruppenknoten deaktiviert, dann wird die Teilszene nicht geladen.

7.2.2 Aufteilung großer Szenen

Mit Hilfe der *ProxyGroup* lassen sich auf einfache Weise große Szenenbeschreibungen unterteilen. Hierbei enthält der Szenengraph eine Vielzahl von *ProxyGroups*, die auf Teilegraphen der Szene verweisen.

Die *ProxyGroup* wird bei der Aufteilung großer Szenen wie eine bedingte Einfügung verwendet. Ein Teilgraph wird nur dann geladen, wenn er sichtbar und die *ProxyGroup* aktiviert ist. Bei der Lastverteilung wird jedem Rechner eine Liste der Knoten des Graphen übergeben, die gezeichnet werden sollen. Alle *ProxyGroups*, die nicht in dieser Liste enthalten sind, werden deaktiviert. Eine Teilszene wird dadurch nur von einem einzelnen Rechner geladen.

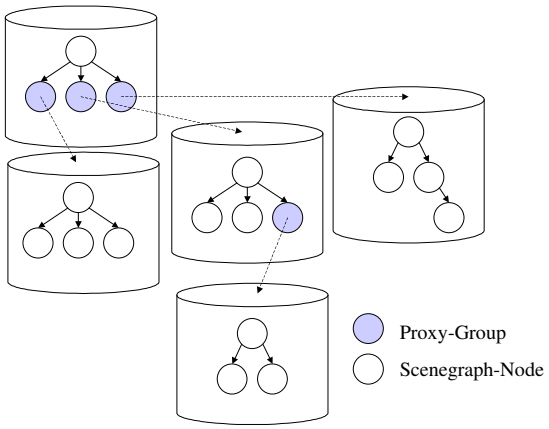

Abbildung 7-10: Zerlegung des Szenengraphen mit Hilfe von *ProxyGroups*

7.2.3 Parallelisierung des Ladevorgangs

Bei der Anzeige großer Szenenbeschreibungen sind häufig sehr lange Ladezeiten erforderlich. Die Ladezeit ist bestimmt durch das Lesen und Interpretieren der Szenenbeschreibung. Das Lesen einer Gigabyte großen Szenenbeschreibung kann mehrere Minuten in Anspruch nehmen [ElS03,RYG04]. Eine wesentliche Beschleunigung durch das parallele Laden unterschiedlicher Teile der Szene von mehreren Rechnern ist nicht zu erwarten, da hierbei der Zugriff auf den File Server einen Flaschenhals darstellt. Der Lesevorgang lässt sich nur dann wesentlich beschleunigen, wenn die Größe der Szenenbeschreibung reduziert wird. Um dies zu erreichen, werden alle Szenenteile in einem sehr kompakten Dateiformat gespeichert und anschließend komprimiert. Im Vergleich zu Szenenbeschreibungen im VRML- oder Inventor-Format lässt sich hiermit die Dateigröße um den Faktor 10 bis 20 reduzieren. Beim Lesen müssen die Teilszenen entkomprimiert werden. Dieser Mehraufwand fällt jedoch kaum ins Gewicht, da die Teilszenen von unterschiedlichen Rechnern parallel entpackt werden.

Der parallele Ladevorgang profitiert auch davon, dass beim Lesen und Interpretieren der Szenenbeschreibung kein einzelner Rechner die komplette Szene bearbeiten muss. Jeder Rechner muss nur einen Teil der Beschreibung interpretieren und in seinen lokalen Szenengraph integrieren. Damit wird es auch möglich, sehr große Szenenbeschreibungen in kurzer Zeit zu visualisieren. Mit Hilfe der *ProxyGroups* ist es möglich, durch eine Szene zu navigieren, während einzelne Teile noch geladen werden.

Das David-Modell aus dem Stanford Scanning Repository mit etwa 50 Millionen Polygonen kann in einem Cluster aus 32 PCs bereits nach 30 Sekunden dargestellt werden. Befinden sich die Daten im Cache der einzelnen Rechner, ist bereits nach zehn Sekunden das erste vollständige Bild zu sehen.

7.2.4 Ein erweitertes Dateiformat für große Szenen

Der oben beschriebene Ansatz liefert in Verbindung mit einer Sort-Last-parallelen Bildberechnung sehr gute Ergebnisse. Es können große Szenen schnell geladen und in interaktiven Bildwiederholraten dargestellt werden. Ein Nachteil besteht jedoch darin, dass eine Szenenbeschreibung in sehr viele kleine Teilszenen unterteilt werden muss. Die Handhabung der hierbei anfallenden vielen Einzeldateien ist für einen Anwender wesentlich aufwendiger als die Handhabung einer einzelnen Szenenbeschreibung. Damit eine Szene auch aus einer einzelnen Beschreibungsdatei parallel geladen werden kann, werden zwei unterschiedliche Ansätze realisiert.

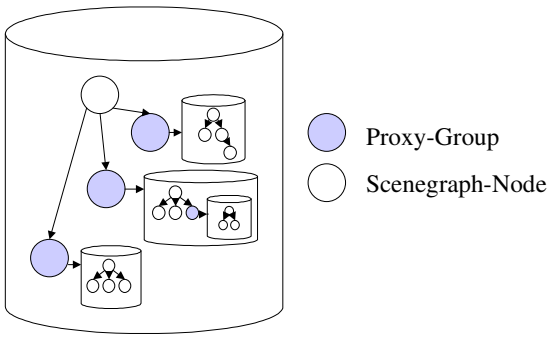

Abbildung 7-11: Eingebettete komprimierte Szenenbeschreibungen

Im ersten Ansatz werden die Teilszenen komprimiert und direkt als Bytefolge in ein Feld der *ProxyGroup* geschrieben. Eine Applikation liest bei diesem Ansatz die vollständige Beschreibung mit allen komprimierten Teilszenen ein. Die Beschreibungen der Teilszenen werden jedoch nicht interpretiert und auch nicht entpackt. Anschließend wird der Szenengraph per Multicast an alle Rechner im Cluster verteilt. Erst dann werden die Teilszenen von jeweils einem Rechner entpackt und interpretiert. Bei diesem Ansatz muss die Client-Applikation mehr Daten lesen. Wie jedoch bereits oben beschrieben, ist der reine Lesevorgang von einem einzelnen File Server durch eine Parallelisierung kaum zu beschleunigen. Meist können wenige große Dateien wesentlich schneller geladen werden als viele kleine. Der Nachteil bei diesem Ansatz besteht darin, dass durch die Einbettung der komprimierten Szenenbeschreibungen in den Szenengraph der Speicherverbrauch zunimmt. Die Größe der ladbaren Szenenbeschreibung ist nicht mehr nur durch den summierten Hauptspeicher aller parallel rechnenden Knoten im Cluster bestimmt, sondern von der Kompressionsrate, mit der die Szene komprimiert werden kann und dem zur Verfügung stehenden Hauptspeicher eines einzelnen Rechners. Wenn die Szene komprimiert in den Hauptspeicher geladen werden kann, dann ermöglicht dieser Ansatz ein sehr effizientes Laden der Szene. Kann auch die komprimierte Szene nicht mehr geladen werden, dann muss der im Folgenden beschriebene Ansatz gewählt werden.

Im zuvor beschriebenen Ansatz werden alle Teilszenen in komprimierter Form in den Szenengraph eingebettet und vollständig von der Client-Applikation geladen. Ein anderer Ansatz besteht darin, alle Teilszenen hintereinander in eine einzelne Datei zu schreiben. Vor jeder Teilszene wird in einem Längenfeld die Größe der Szenenbeschreibung vermerkt. Mit dieser Information ist es möglich, sehr schnell einzelne Teilszenen aus der zusammengesetzten Datei zu extrahieren. Der

Hauptszenengraph enthält wieder *ProxyGroups*, die nun jedoch keinen Verweis auf eine Datei und keine komprimierte Szenenbeschreibung besitzen, sondern einen Verweis auf den Anfang der Teilszene in der zusammengesetzten Datei. Auf dem Client wird lediglich die erste Szene der Datei geladen. Die restlichen Teile werden erst nach der Verteilung des Szenengraphen von den Rechnern im Cluster nachgeladen. Die Geschwindigkeit entspricht der des ursprünglichen Proxy-Konzeptes mit einem Verweis auf eine Datei. Der Vorteil liegt lediglich darin, dass alle Teilszenen in einer einzigen Datei zusammengefasst sind.

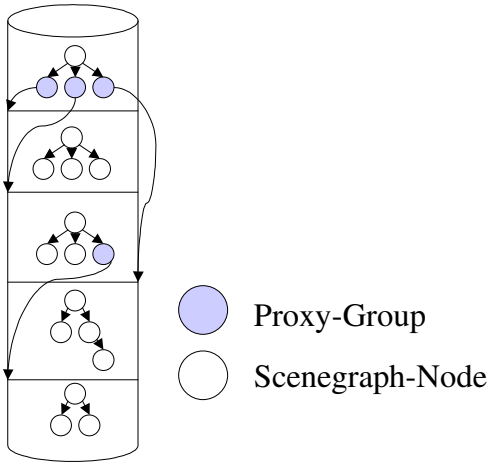

Abbildung 7-12: Zusammengesetzte Szenenbeschreibungsdateien

7.3 Integration in das VR-System Avalon

Die Verwendbarkeit der Ergebnisse dieser Arbeit soll an der Integration in das VR/AR-System Avalon aufgezeigt werden. Avalon ist ein VRML97- und X3D-basiertes VR-System. Es dient als Laufzeitumgebung für interaktive 3D-Anwendungen [BDR04]. Avalon wurde mit dem Ziel entwickelt, eine erweiterbare, komponentenbasierte VRML-Laufzeitumgebung bereitzustellen. Wie jedes VRML-basierte System besteht es aus einer Ansammlung von Knoten, die wiederum aus einzelnen Feldern bestehen. Der Datenaustausch zwischen Feldern unterschiedlicher Knoten erfolgt mit Hilfe so genannter Routes. Hierbei verbindet eine Route zwei Felder, wobei Feldänderungen des Quellfeldes automatisch an das Zielfeld einer Route weitergeleitet werden. In Avalon werden mit Hilfe dieses Konzeptes zwei separate Graphen aufgebaut. Ein Graph enthält die vollständige Beschreibung der Szene mit allen interaktiven Elementen. Der zweite Graph dient der Beschreibung der Laufzeitumgebung. Diese strikte Trennung erlaubt es, eine Szene unverändert mit unterschiedlichen Interaktionsgeräten und grafischen Ausgabegeräten darzustellen. Die beiden Graphen werden in Avalon als *Engine* und *Szene* bezeichnet. Da die Konfiguration eines Clusters als Teil der Laufzeitumgebung angesehen werden kann, wird in dieser Arbeit der Graph der *Engine* für eine nahtlose Integration des Cluster-Supports herangezogen.

Avalon verwendet zur Bildberechnung das Szenengraphensystem OpenSG. Hierfür werden die im VRML-Standard vorgesehenen Knoten auf OpenSG abgebildet. Diese Abbildung kann für die Unterstützung der Bildberechnung in einem Cluster unverändert beibehalten werden. Es müssen lediglich die Komponenten in Avalon aufgenommen werden, die für die Konfiguration des

grafischen Kontexts erforderlich sind. Alle entwickelten Mechanismen zur Konfiguration von unterschiedlichen Projektionssystemen und parallelen Bildberechnungsverfahren werden so in Avalon integriert, dass sie auf der Basis des Field-und-Route-Konzeptes von VRML verwendet werden können.

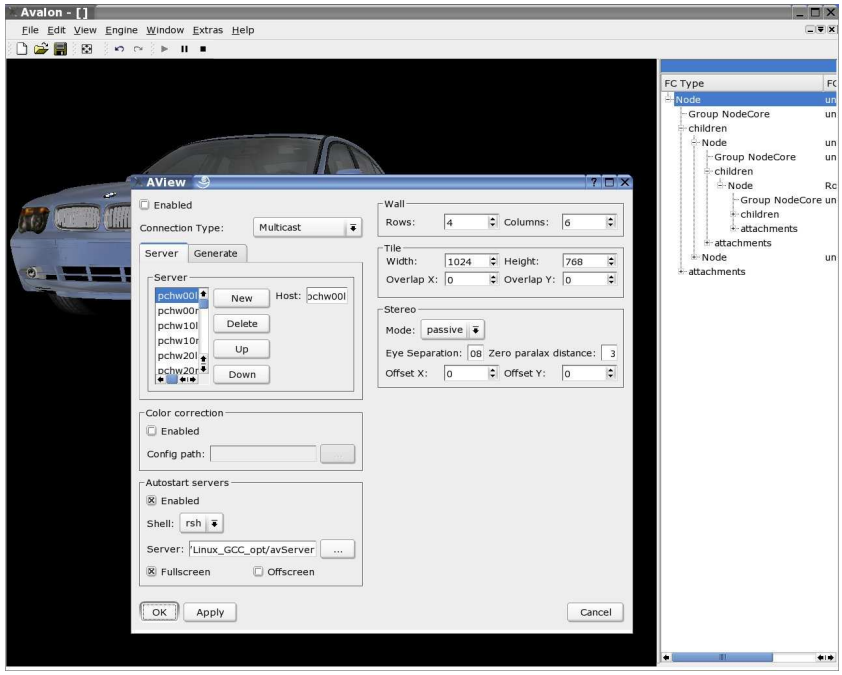

Abbildung 7-13: Cluster-Konfiguration in Avalon

Mit den in dieser Arbeit entwickelten Techniken lassen sich beliebige Projektionssysteme betreiben. Die große Flexibilität erhöht jedoch den Arbeitsaufwand bei der Konfiguration der Cluster-Umgebung. Um dies zu vereinfachen, wird Avalon um einfache Konfigurationsdialoge für Standard-Displays wie z.B. gekachelte Projektionen, stereoskopische Displays und Cave-Projektionen erweitert. Damit ist es Anwendern möglich, bereits nach einer kurzen Einarbeitungszeit ein Projektionssystem mit Avalon zu betreiben. Abbildung 7-13 zeigt das Konfigurationsmenü für eine gekachelte Projektion.

Durch die separate Verwaltung eines OpenSG-Szenengraphen und eines VRML-Szenengraphen müssen in Avalon nur solche Teile des VRML/X3D-Graphen im Cluster verteilt werden, die für die Darstellung relevant sind.

Durch die erfolgreiche Integration des Cluster-Supports in Avalon ist die verteilte Bildberechnung auf einfache Weise möglich. Da Avalon die sehr verbreiteten Dateiformate VRML und X3D unterstützt, können Anwendungen, die auf diesen Formaten basieren, sehr einfach in einer Cluster-Umgebung visualisiert werden.

7.4 Zusammenfassung

In diesem Kapitel wurde gezeigt, dass es mit den vorher beschriebenen Techniken möglich ist, alle Projektionssysteme anzusteuern, die aus mehreren planaren Einzelprojektionen zusammengesetzt sind. Basierend auf den Ergebnissen dieser Arbeit wurden mittlerweile viele unterschiedliche Projektionssysteme mit Cluster-Systemen betrieben. Dies umfasst einfache Stereo-Projektionen, Caves mit aktivem und passivem Stereo, Panoramaprojektionen, gekachelte Projektionen bis hin zur HEyeWall mit 48 Projektoren. Durch die Unterstützung der Farbanpassung zwischen den Projektoren und der geometrischen Korrektur der projizierten Bilder lassen sich mit preiswerten Projektoren hochwertige Projektionssysteme aufbauen.

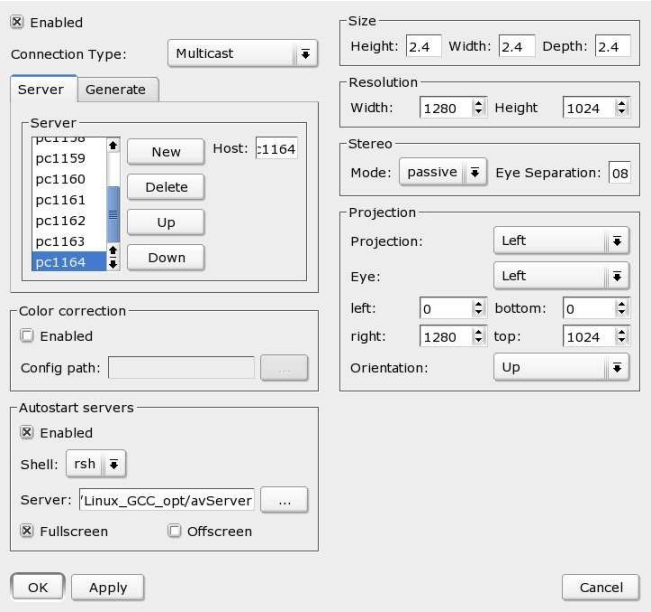

Abbildung 7-14: Konfigurationsdialog für eine Cave

Es wurde gezeigt, wie auch große Szenen interaktiv angezeigt werden können. Ebenso wie die Bildberechnung kann auch der Ladevorgang auf viele Rechner in einem Cluster verteilt werden. Die Größe der darstellbaren Szenen ist lediglich durch die Summe des Hauptspeichers aller Rechner im Cluster beschränkt. Durch die einfache Handhabung ist es möglich, teilweise auf die sonst übliche zeitaufwendige Datenaufbereitung für VR-Systeme zu verzichten. Meist werden aus CAD-Systemen sehr komplexe Modelle exportiert, die dann für eine VR-Anwendung vereinfacht werden müssen. Entfällt die Notwendigkeit der Aufbereitung, so können CAD-Daten wesentlich schneller in VR-Systemen begutachtet werden

8 Zusammenfassung und Ausblick

Ziel der vorliegenden Arbeit war es, ein Szenengraphensystem für die parallele Bildberechnung in einem Cluster zu konzeptionieren, zu realisieren und zu evaluieren. Dabei ist ein hochperformantes System entstanden, mit dessen Hilfe PC-Cluster sehr einfach und effizient für die Berechnung dreidimensionaler Bilder in interaktiven Anwendungen eingesetzt werden können.

Die Unterstützung von Clustern wurde dabei vollständig in ein Szenengraphensystem integriert. Damit wurde es möglich, die Bildberechnung in einem Cluster vollständig von der Applikationsschnittstelle zu trennen. Das Basissystem wurde als offenes Rahmensystem entworfen, um eine einfache Integration neuer Hardware und neuer Bildberechnungsalgorithmen zu ermöglichen.

Die Kommunikation zwischen den Knoten eines Clusters stellt das größte Problem bei der Umsetzung von Algorithmen von Einzel- oder Multiprozessorsystemen auf Cluster-Systeme dar. Aus diesem Grund wurde ein effizientes und erweiterbares Kommunikations-Framework entworfen, mit dessen Hilfe unterschiedliche Cluster-Netzwerke in das System integriert werden können. Auf der Basis von Multicast über Ethernet wurde ein neues Protokoll entwickelt, um Daten sehr schnell und zuverlässig an eine große Zahl von Empfängern zu versenden. Damit ist es möglich, auch in dynamischen Szenen mit Partikelsystemen oder animierter Geometrie hohe Bildwiederholraten zu erreichen.

Bei der parallelen Bildberechnung kann entweder der Bildraum oder die Szene unterteilt und parallel berechnet werden. Beide Ansätze bieten Vor- und Nachteile, weshalb für beide Ansätze neue Verfahren entwickelt wurden, die auf die besonderen Anforderungen von Cluster-Systemen eingehen. Es wurde ein neues Verfahren entwickelt, mit dessen Hilfe sich beliebige Projektionssysteme betreiben lassen, die aus mehreren Einzelprojektionen zusammengesetzt sind. Hierbei kann der Berechnungsaufwand zwischen den einzelnen Projektionen und auf zusätzliche Rechner verteilt werden. Das Verfahren ist nicht auf eine vordefinierte Anzahl von Konfigurationen wie z.B. Cave, Stereo-Projektion oder Tiled-Wall beschränkt, sondern lässt sich auf alle Projektionssysteme anwenden, die aus planaren Einzelprojektionen zusammengesetzt sind.

Die Sort-Last-parallele Bildberechnung skaliert mit steigender Zahl parallel rechnender Einheiten wesentlich besser als die Sort-First-parallele Bildberechnung. Das Problem bei Sort-Last besteht jedoch in der großen Datenmenge, die beim Zusammenfügen des fertigen Bildes zwischen den Rechnern ausgetauscht werden muss. Um auch mit Sort-Last interaktive Bildwiederholraten zu erreichen, wurde ein neues Bildkompositionsverfahren entwickelt und mit bestehenden Verfahren verglichen. Das neu entwickelte Sorted-Pipeline-Verfahren erfordert eine wesentlich geringere Bandbreite und ist daher besser für den Einsatz in Cluster-Systemen geeignet. In einem Cluster, bestehend aus 90 PCs, wurde ein Durchsatz von 2,4 Milliarden Polygonen/s erreicht.

In einem Anwendungsbeispiel wurde gezeigt, dass mit den entwickelten Techniken beliebige Projektionssysteme konfiguriert und mit Hilfe einer Farb- und Projektionskorrektur auch günstige Projektoren für hochwertige Projektionssysteme eingesetzt werden können. Es wurde gezeigt, dass mit Hilfe des entwickelten Systems in einem Cluster sehr große Modelle parallel geladen und in interaktiven Bildwiederholraten dargestellt werden können. Da der Szenengraph meist Teil eines AR- oder VR-Systems ist, wurde die Integration in das VR-System Avalon vorgenommen. Auf der Basis dieser Integration wurde eine interaktive Konfigurationskomponente für Cluster-Systeme entwickelt.

Alle in dieser Arbeit entwickelten Konzepte und Verfahren sind Bestandteil von OpenSG. OpenSG ist frei zugänglich und wurde in den letzen Jahren für viele Projekte an Forschungseinrichtungen eingesetzt. Die Verwendbarkeit der Cluster-Komponenten in OpenSG wurde vielfältig unter Beweis gestellt. Es wurden sowohl Anwendungen auf dem bestehenden Framework entwickelt als auch neue Verfahren in das Framework integriert. OpenSG ist heute das einzige Szenengraphensystem mit einer breiten Unterstützung für Cluster-Systeme.

Das in dieser Arbeit entwickelte Framework ist auf Erweiterbarkeit ausgelegt. Es ist zu erwarten, dass das bestehende Framework um die Unterstützung neuer Hardware zur beschleunigten Bildberechnung in Clustern erweitert wird. Um die Bildberechnung weiter zu beschleunigen, ist es wichtig, neue Cluster-Netzwerke in das Kommunikations-Framework zu integrieren und weitere Hardware zur Bildkomposition anzubinden.

In Kapitel 6 wurde ein einfacher Ansatz zur Darstellung transparenter Flächen im Zusammenhang mit der Sort-Last-parallelen Bildberechnung beschrieben. Die Informationen, die bei der Sorted-Pipeline-Bildkomposition anfallen, können als Basis für eine globale Sortierung transparenter Flächen herangezogen werden. Darauf aufbauend könnte eine sehr effiziente parallele Darstellung transparenter Flächen entwickelt werden.

Diese Arbeit konzentriert sich auf die Darstellung polygonaler Modelle. In zukünftigen Arbeiten können viele der entwickelten Verfahren auch für die Parallelisierung der Darstellung von Volumen-Daten erweitert werden.

AhP98 James Ahrens and James Painter. Efficient sort-last rendering using compression-based image compositing. In: Second Eurographics Workshop on Parallel Graphics and Visualisation, 1998.

BBF00 W. Blanke, C. Bajaj, D. Fussell, and X. Zhang. The metabuffer: A scalable multiresolution multidisplay 3-d graphics system using commodity rendering engines. technical report tr2000-16. Technical report, University of Texas at Austin, 2000.

BCD04 A. Baczyk, A. Carbone, J.P. Dufey, D. Galli, B. Jost, U. Marconi, N. Neufeld, and G. Peco V. Vagnoni. Reliability of datagram transmission on gigabit ethernet at full link load. Technical Report LHCB-2004-030 DAQ, LHCb Bologna, 2004.

BCF95 N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. Su. Myrinet. A gigabit-per-second local area network. In: IEEE Micro, Vol. 15, pages 29--36, 1995.

BDR04 Johannes Behr, Patrick Dähne, and Marcus Roth. Utilizing x3d for immersive environments. In: Ninth international conference on 3D Web technology, Monterey, California, pages 71--78, 2004.

BEF02 Johannes Behr, Peter Eschler, Torsten Froehlich, Christian Knoepfle, Bernd Lutz, Stefan Mueller, and Marcus Roth. Cybernarium days 2002 - a public experience of virtual and augmented worlds. In: First International Symposium on Cyber Worlds. Proceedings 2002 : Theory and Practices, pages 553--560, 2002.

BaL98 Amnon Barak and Oren La'adan. The MOSIX multicomputer operating system for high performance cluster computing. Future Generation Computer Systems, 13(4--5):361--372, 1998.

BSS95 Donald J. Becker, Thomas Sterling, Daniel Savarese, John E. Dorband, Udaya A. Ranawak, and Charles V. Packer. Beowolf: a parallel workstation for scientific computation. In: Proceedings of International Conference on Parallel Processing, 1995.

CoB97 M. Cox and N. Bhandari. Architectural implications of hardwareaccelerated bucket rendering on the pc. In: 1997 Siggraph/Eurographics Workshop on Graphics Hardware, Los Angeles, CA, August 1997, 1997.

CCC01 Yuqun Chen, Han Chen, Douglas W. Clark, Zhiyan Liu, Grant Wallace, and Kai Li. Software environments for cluster-based display systems. In: IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001.

CCF01 Han Chen, Yuqun Chen, Adam Finkelstein, Thomas Funkhouser, Kai Li, Zhiyan Liu, Rudrajit Samanta, and Grant Wallace. Data distribution strategies for high-resolution displays. Computers and Graphics, 25(5), 2001.

CaD92 Stephen Casner and Stephen Deering. First ietf internet audiocast. In: ACM Computer Communication Review, 1992. Vol. 22, No. 3 (July 1992), pp. 92--97. Earlier version is appeared in ConneXions: The Interoperability Report, Vol.6, No.6 (June 1992).

CoH92 Michael Cox and Pat Hanrahan. Depth complexity in object parallel graphics architectures. In: Proceedings of the Seventh Workshop on Graphics Hardware, Eurographics '92, Cambridge, England, pages 204--222, September 1992.

CoH94 M. Cox and P. Hanrahan. A distributed snooping algorithm for pixel merging. In: IEEE Parallel and Distributed Technology: Systems and Applications, Volume: 2 Issue: 2, pages 30--36, 1994.

Che96 X. Chen. Gigabit network multicast protocols, Ph.D. Dissertation, University of California, Santa Barbara (June 1996), 1996.

CKS02 Wagner Correa, James Klosowski, and Claudio Silva. Out-of-core sort-first parallel rendering. In: Eurographics Workshop on Parallel Grahpics and Visualisation, pages 9--10, 2002.

CSD93 Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A. DeFanti. Surround-screen projection-based virtual reality: The design and implementation of the cave. In: SIGGRAPH 93, pages 135--142, 1993.

CSW02 H. Chen, R. Sukthankar, G. Wallace, and K. Li. Scalable alignment of large-format multi-projector displays using camera homography trees, In: Proceedings of IEEE Visualization 2002.

CrO91 Thomas W. Crockett and Tobias Orloff. A parallel rendering algorithm for MIMD architectures. ICASE Report 91-3, 1991.

Cro95 T. W. Crockett. Parallel rendering. Technical report, NASA, 1995.

CSI98 Milton Chen, Gordon Stoll, Homan Igehy, Kekoa Proudfoot, , and Pat Hanrahan. Simple models of the impact of overlap in bucket rendering. In: 1998 SIGGRAPH Eurographics Workshop on Graphics Hardware, pages 105--112, August 1998.

Bak00 Mark Baker (ed). Cluster computing white paper, IEEE Task Force on Cluster Computing, 2000.

EIH00 Matthew Eldridge, Homan Igehy, and Patrick M. Hanrahan. Pomegranate: A fully scalable graphics architecture. In: SIGGRAPH 2000, pages 443--454, 2000.

Ell94 D. Ellsworth. A new algorithm for interactive graphics on multicomputers. In: IEEE Computer Graphics and Applications, Vol. 14, No. 4, pages 33--40, 1994.

Ell96 D. Ellsworth. Polygon Rendering for Interactive Visualization on Multicomputers. PhD thesis, University of North Carolina at Chapel Hill, 1997.

ElS03 Daniel Ellard and Margo Seltzer. Nfs tricks and benchmarking traps. In: USENIX Annual Technical Conference, pages 1001--114, 2003.

FJL97 S. Floyd, V. Jacobson, C. Liu, S. McCanne, , and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In: IEEE/ACM Transactions on Networking, December 1997, Volume 5, Number 6, pages 784--803, 1997.

FUL00 Thomas Funkhouser and Kai Li. Large format displays. In: IEEE Computer Graphics and Applications , 25 (4), pages 20--21, 2000.

MPI94 Message Passing Interface Forum. MPI: A message-passing interface standard, 1994.

FrR00 Torsten Fröhlich and Marcus Roth. Integration of multidimensional interaction devices in real-time computer graphics applications. Computer Graphics Forum, 19(3):313--320, August 2000.

FuS93 T. A. Funkhouser and C. H. Sequin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. Computer Graphics Annual Conference Series, pages 247--254, 1993.

Fun96 T. Funkhouser. Network topologies for scalable multi-user virtual environments. Proceedings of VRAIS'96, Santa Clara CA, pages 222--229, 1996.

GBD94 A. Geist, A. Beguelin, Jack Dongarra, W. Jiang, R. Manchek, and V. Sunderam. PVM Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing. MIT Press, Cambridge, Mass., 1994.

HEB01 Greg Humphreys, Matthew Eldridge, Ian Buck, Gordon Stoll, Matthew Everett, and Pat Hanrahan. Wiregl: A scalable graphics system for clusters. In: Proceedings of ACM SIGGRAPH 2001, Computer Graphics Proceedings, Annual Conference Series, pages 129--140, August 2001.

Hei94 R. Heiland. Object-oriented parallel polygon rendering. In: ACM Graphics and Visualization Conference, pages 19--26, 1994.

HHN02 Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: A stream processing framework for interactive graphics on clusters, Proceedings of ACM SIGGRAPH 2002.

HeM99 Alan Heirich and Laurent Moll. Scalable distributed visualization using off-the-shelf components. In: IEEE Parallel Visualization and Graphics Symposium, pages 55--59, October 1999.

HSF99 Gerd Hesina, Dieter Schmalstieg, Anton Fuhrmann, and Werner Purgathofer. Distributed open inventor: A practical approach to distributed 3d graphics. In: Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST'99), pages 74--81, 1999.

Inf02 Infiniband specification 1.1, InfiniBand Trade Association, November 2002.

Intel03 Interrupt moderation using intel gigabit ethernet controllers application note, Technical Report Intel AP-450, 2003.

JGJ00 J. Jannotti, D. K. Gifford, and K. L. Johnson. Overcast: Reliable multicasting with an overlay network. In: 4th Symposium on Operating Systems Design and Implementation, 2000.

KRS03 Christian Kurmann and Felix Rauch andomas M. Stricker. Cost/performance tradeoffs in network interconnects for clusters of commodity pcs. In: Workshop on Communication Architecture for Clusters. In conjunction with International Parallel and Distributed Processing Symposium (IPDPS '03), 2003.

KAO98 T. Kurc, C. Aykanat, and B. Ozguc. Object-space parallel polygon rendering on hyper-cubes. Computers and Graphics, 22(4):487--503, 1998.

KKH03 P. D. Kirchner, J. T. Klosowski, P. Hochschild, and R. Swetz. Scalable visualization using a network-attached video framebuffer. Computers & Graphics, 27(5):669--680, 2003.

KRK03 Wolfram Kresse, Dirk Reiners, and Christian Knoepfle. Color consistency for digital multi-projector stereo display systems: the heyewall and the digital cave. In: Proceedings of the workshop on Virtual environments 2003, pages 271

KoZ96 Alex Koifman and Stephen Zabele. Ramp: A reliable adaptive multicast protocol. In: IEEE INFOCOMM96, pages 1442--1451, 1996.

Law96 Asish Law. Expoloiting Cohernecy in Parallel Algorithms for Volume Rendering. PhD thesis, Ohio State University, 1996.

LCC00 K. Li, H. Chen, Y. Chen, D. Clark, P. Cook, S. Daminakis, G. Essl, A. Finkelstein, T. Funkhouser, A. Klein, Z. Liu, E. Praun, R. Samanta, B. Shedd, J. Singh, G. Tzanetakis, and J. Zheng. Building and using a scalable display wall system. In: Computer Graphics and Applications, 20(4), 2000.

LDY01 R. Lane, S. Daniels, and X. Yuan. An empirical study of reliable multicast protocols over ethernet--connected networks. Technical Report TR-010503, CSD, Florida State University, 2001.

LaL94 Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. Computer Graphics, 28(Annual Conference Series):451--458, 1994.

LMS01 Santiago Lombeyda, Laurent Moll, Mark Shand, David Breen, and Alan Heirich. Scalable interactive volume rendering using off-the-shelf components. In: IEEE 2001 Symposium on Parallel and Large-Data Visu alization and Graphics, pages 115--121, 2001.

LRN96 Tong-Yee Lee, C. S. Raghavendra, and J. B. Nicholas. Image composition schemes for sort-last polygon rendering on 2d mesh multicomputers. In: IEEE Transactions on Visualization and Computer Graphics. 2(3), pages 202--217, 1996.

Orad04 Orad Hi-Tec Systems Ltd. Dvg technical presentation, http://www.orad.co.il/visual.htm, 2004.

MaS04 A. Majumder and R. Stevens. Color nonuniformity in projection based displays: Analysis and solutions, In: IEEE Transactions on Visualization and Computer Graphics March/April 2004 – Vol 10 Nr 2 pages 177--188, 2004.

MAM95 G. Mandyam, N. Ahmed, and N. Magotra. A dct-based scheme for lossless image compression, IS&T/SPIE Electronic Imaging Conference. San Jose, CA. February, 1995

MAM97 G. Mandyam, N. Ahmed, and N. Magotra. Lossless image compression using the discrete cosine transform, Proceedings of SIGGRAPH 97, pages 293—302, 1997.

MeB76 Robert M. Metcalfe and David R. Boggs. Ethernet: Distributed packet switching for local computer networks. Communications of the ACM, 19(5):395--404, 1976.

MBD97 John S. Montrym, Daniel R. Baum, David L. Dignam, and Christopher J. Migdal. Infinitereality: A real-time graphics system. In: Proceedings of SIGGRAPH 97, Computer Graphics Proceedings, Annual Conference Series, pages 293--302, August 1997.

MiC98 Tulika Mitra and Tzi cker Chiueh. Implementation and evaluation of the parallel mesa library. In: IEEE International Conference on Parallel and Distributed Systems, pages 84--91, 1998.

MCE94 Steve Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A sorting classification of parallel rendering. In: IEEE Computer Graphics and Applications, pages 23--31, 1994.

MEP92 Steven Molnar, John Eyles, and John Poulton. PixelFlow: High-speed rendering using image composition. Computer Graphics, 26(2):231--240, 1992.

MHS99 Laurent Moll, Alan Heirich, and Mark Shand. Sepia: scalable 3D compositing using PCI Pamette. In: Kenneth L. Pocek and Jeffrey Arnold, editors, IEEE Symposium on FPGAs for Custom Computing Machines, pages 146--155, Los Alamitos, CA, 1999. IEEE Computer Society Press.

Mol91 Steven Molnar. Image-Composition Architectures for Real-time Image Generation. PhD thesis, University of North Carolina, 1991.

MPH94 Kwan-Liu Ma, James S. Painter, Charles D. Hansen, and Michael F. Krogh. Parallel volume rendering using binary-swap compositing. IEEE Computer Graphics and Applications, 14(4):59--68, 1994.

MRW99 P.K. Mckinley, R. T. Rao, and R. F. Wright. H-rmc: A hybrid reliable multicast protocol for the linux kernel. In: Proceedings of IEEE SC99: High Performance Networking and Computing, 1999.

Mue95 C. Mueller. The sort-first architecture for high-performance graphics. In: Symposium on Interactive 3D Graphics (Monterey, California, April 9-12, 1995), pages 75--84, 1995.

Mue00 C. Mueller. The Sort-First Architecture for RealTime Image Generation. PhD thesis, CS UNC Chapel Hill, 2000.

MWP01 K. Moreland, B. Wylie, and C. Pavlakos. Sort-last parallel rendering for viewing extremely large data sets on tile displays. In: IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics, pages 85--154, 2001.

MaZ97 M. R. Macedonia and M. J. Zyda. A taxonomy for networked virtual environments. In: IEEE Multimedia, 4(1), pages 48--56, 1997.

MZP94 M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz. NPSNET: A network software architecture for large-scale virtual environment. Presence: Teleoperators and Virtual Environments, 3(4):265--287, 1994.

Neu93 U. Neumann. Parallel volume-rendering algorithm performance on mesh-connected multicomputers. In: Parallel Rendering Symposium, pages 97--104, 1993.

NgZ00 Thu Nguyen and John Zahorjan. Image layer decomposition for distributed rendering on nows. In: 2000 International Parallel and Distributed Processing Symposium, Cancun, Mexico, May 1-5, 2000.

HP04 High Performance Technical Computing Division of Hewlett-Packard Corporation. Presentation: High end visualization for performance computing, http://www.hp.com/techservers/hpccn/sci_vis/resources.html, 2004

PeJ01 Perrine, K. A. and Jones, D. R. 2001. Parallel graphics and interactivity with the scaleable graphics engine. In: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing. 2001.

PFH02 Fabrizio Petrini, Wu chun Feng, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg. The Quadrics Network: High Performance Clustering Technology. IEEE Micro, 22(1):46--57, January-February 2002.

PLG99 Stefan Petri, Gunther Lustig, Claus Grewe, Rainer Hagenau, Wolfgang Obeloeer, and Marcel Boosten. Performance comparison of different high-speed networks. In: SCI-Europe '99, Toulouse, 1999.

PSL97 S. Paul, K. K. Sabnani, J. C. Lin, and S. Bhattacharyya. Reliable multicast transport protocol (rmtp). IEEE Journal on Selected Areas in Communications, 15(3):407--421, 1997.

Rei02 Dirk Reiners. OpenSG. PhD thesis, TU Darmstadt, Fachbereich Informatik, 2002.

RoH94 J. Rohlf and J. Helman. Iris performer: A high performance multiprocessing toolkit for real-time 3d graphics. In: Proceedings of SIGGRAPH 94, ACM SIGGRAPH Annual Conference Series, pages 381--395, 1994.

RaH96 S. Rak and D. Van Hook. Evaluation of grid-based relevance filtering for multicast group assignment. In: 14th Workshop on Standards for the Interoperability of Distributed Simulations, pages 739--747, 1996.

RaH01 Zoran Radović and Erik Hagersten. Dszoom – low latency software-based shared memory. Technical Report 2001:03, Parallel and Scientific Computing Institute (PSCI), Sweden, April 2001.

ROT02 Marcus Roth. Integration paralleler Rendering-Verfahren fuer lose gekoppelte Systeme in OpenSG. In: OpenSG Symposium, Januar 2002, 2002.

RRV03 Marcus Roth, Dirk Reiners, Gerrit Voss, and Johannes Behr. Flexible and opaque clustering support for scene graph systems. In: VR-Cluster '03 Workshop on Commodity Clusters for Virtual Reality 2003, pages 22--26, 2003.

RaS99 C. R. Ramakrishnan and Claudio T. Silva. Optimal processor allocation for sort-last compositing under bsp-tree ordering. In: SPIE Electronic Imaging, Visual Data Exploration and Analysis IV, January 1999.

RVB02 D. Reiners, G. Voss, and J. Behr. Opensg: Basic concepts. In: OpenSG Forum, 2002.

RVR04 Marcus Roth, Gerrit Voss, and Dirk Reiners. Multi-threading and clustering for scene graph systems. Computers and Graphics, 28(1):63--66, 2004.

RYG04 Peter Radkov, Li Yin, Pawan Goyal, Prasenjit Sarkar, and Prashant Shenoy. A performance comparison of nfs and iscsi for ip-networked storage. In: Proceedings of the Usenix Conference on File and Storage Technologies (FAST 04), San Francisco, CA, 2004.

SEP01 Gordon Stoll, Matthew Eldridge, Dan Patterson, Art Webb, Steven Berman, Richard Levy, Chris Caywood, Milton Taveira, Stephen Hunt, and Pat Hanrahan. Lightning-2: A high-performance display subsystem for pc clusters, Computer Graphics (SIGGRAPH 2001 Proceedings), 2001.

SaF98 Rudrajit Samanta and Thomas Funkhouse. Dynamic algorithms for sorting primitives among screen-space tiles in a parallel rendering system. Technical report, Department of Computer Science, Princeton University, October 1998.

SFL01 R. Samanta, T. Funkhouser, and Kai Li. Parallel rendering with k-way replication. In: IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics, pages 75--153, 2001.

SFL00 Rudrajit Samanta, Thomas Funkhouser, Kai Li, and Jaswinder Pal Singh. Hybrid sort-first and sort-last parallel rendering with a cluster of pcs. In: 2000 SIGGRAPH / Eurographics Workshop on Graphics Hardware, pages 97--108, 2000.

SiJ02 Steven J. Sistare and Christopher J. Jackson. Ultra-high performance communication with mpi and the sun fire(tm) link interconnect. In: SC2002 Baltimore, 2002.

SZF99 Rudrajit Samanta, Jiannan Zheng, Thomas Funkhouser, Kai Li, Jaswinder, and Pal Singh. Load balancing for multi-projector rendering systems. In: 1999 SIGGRAPH / Eurographics Workshop on Graphics Hardware. pp. 107-116, 1999, 1999.

VBR02 G. Voss, J. Behr, D. Reiners, and M. Roth. A multi-thread safe foundation for scenegraphs and its extension to clusters. In: Fourth Eurographics Workshop on parallel Graphics and Visualization, 2002.

Whe85 Daniel S. Whelan. A multiprocessor Architecture for real-time computer animation. PhD thesis, California institute for Technology, 1984.

Whi94 Scott Whitman. Dynamic load balancing for parallel polygon rendering. In: IEEE Computer Graphics and Applications, Vol. 14, No. 4, pages 41--48, 1994.

WMK94 B. Whetten, T. Montgomery, and S. Kaplan. A high performance totally ordered multicast protocol. In: Proceedings of the International Workshop of Theory and Practice in Distributed Systems, Dagstuhl Castle, Germany, pages 33--57, 1994.

CMM96 P.M. Melliar-Smith X. Chen, L.E. Moser. Flow control techniques for multicasting in gigabit networks. In: In: Proceedings of International Conference on Network Protocols, pages 553--560, 1996.

YCC99 Don-Lin Yang, Jen-Chih Yu, Yeh-Ching Chung, and Feng Chia University. Efficient compositing methods for the sort-last-sparse parallel volume rendering system on distributed memory multicomputers. In: International Conference on Parallel Processing, pages 200--207, 1999.

ZBB01 X. Zhang, C. Bajaj, W. Blanke, and D. Fussell. Scalable isosurface visualization of massive datasets on cots clusters. In: IEEE Symposium on Parallel and Large-Data Visualization and Graphics, 2001.

ZHC00 Bob Zeleznik, Loring Holden, Michael Capps, Howard Abrams, and Tim Miller. Scene-Graph-As-Bus: Collaboration between heterogeneous stand-alone 3-D graphical applications. Computer Graphics Forum, 19(3):91--98, 2000.

Lebenslauf

Persönliche Daten

Name:	Marcus Roth
Geburtsdatum:	8. März 1967
Geburtsort:	Kaiserslautern
Familienstand:	Verheiratet, eine Tochter

Schulausbildung:

1973 - 1977	Grundschule Neuwied
1977 - 1978	Hauptschule Mannheim
1978 - 1983	Realschule Bad Dürkheim: Erwerb der Mittleren Reife
1990 - 1992	Telekolleg II: Erwerb der Fachhochschulreife

Berufsaufbildung und Studium:

1983 - 1986	Ausbildung zum Datenverarbeitungskaufmann
1993 - 1997	Studium der Informatik mit Schwerpunkt Technik an der FH Mannheim

Berufstätigkeit:

1986 - 1993	Anwendungsprogrammierer bei der Firma ista GmbH in Mannheim
1997 - 2000	Anwendungsinformatiker bei der Dresdner-Bank AG, Frankfurt
seit 2000	Wissenschaftlicher Mitarbeiter im Zentrum für graphische Datenverarbeitung e.V. in Darmstadt

Veröffentlichungen

- M. Roth. Entwicklung eines Client-Server Systems zur Handhabung multidimensionaler Interaktionsgeräte in 3D Echtzeit-Grafiksystemen, 1997. Diplomarbeit FH Mannheim
- Torsten Fröhlich and Marcus Roth. Integration of multidimensional interaction devices in real-time computer graphics applications. Computer Graphics Forum, 19(3):313--320, August 2000.
- Johannes Behr, Peter Eschler, Torsten Froehlich, Christian Knoepfle, Bernd Lutz, Stefan Mueller, and Marcus Roth. Cybernarium days 2002 – a public experience of virtual and augmented worlds. In First International Symposium on Cyber Worlds. Proceedings 2002: Theory and Practices, pages 553--560, 2002.
- Marcus Roth. Integration paralleler Rendering-Verfahren für lose gekoppelte Systeme in OpenSG. OpenSG Symposium, Januar 2002.
- G. Voss, J. Behr, D. Reiners, and M. Roth. A multi-thread safe foundation for scenegraphs and its extension to clusters. In Fourth Eurographics Workshop on parallel Graphics and Visualization, 2002.
- Marcus Roth, Dirk Reiners, Gerrit Voss, and Johannes Behr. Flexible and opaque clustering support for scene graph systems. In VR-Cluster '03 Workshop on Commodity Clusters for Virtual Reality 2003, pages 22--26, 2003.
- Marcus Roth, Gerrit Voss, and Dirk Reiners. Multi-threading and clustering for scene graph systems. Computers and Graphics, 28(1):63--66, 2004.
- Johannes Behr, Patrick Dähne, and Marcus Roth. Utilizing x3d for immersive environments. In Ninth international conference on 3D Web technology, Monterey, California, pages 71--78, 2004.
- Roth, Marcus, Patrick Riess, Dirk Reiners. Load Balancing on Cluster-Based Multi Projector Display Systems. 14-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2006. WSCG 2006.